Journal of Machine Learning Research 1 (2020) 1-4 Submitted 03/20; Published -

Oríon: Asynchronous Distributed Hyperparameter Optimization

Xavier Bouthillier XAVIER.BOUTHILLIER@UMONTREAL.CA
Christos Tsirigotis CHRISTOS.TSIRIGOTIS@UMONTREAL.CA
Pierre Delaunay PIERRE@DELAUNAY.IO
Thomas Schweizer THOMAS.SCHWEIZER@MILA.QUEBEC
François Corneau-Tremblay FRANCOIS.CORNEAU-TREMBLAY@UMONTREAL.CA
Fabrice Normandin NORMANDF@MILA.QUEBEC
Nadhir Hassen NADHIR.HASSEN@MILA.QUEBEC
Reyhane Askari Hemmat REYHANE.ASKARI.HEMMAT@UMONTREAL.CA
Michael Noukhovitch MICHAEL.NOUKHOVITCH@UMONTREAL.CA
Pascal Lamblin PASCAL.LAMBLIN@GMAIL.COM
Frédéric Bastien BASTIENF@IRO.UMONTREAL.CA
Frédéric Osterrath FREDERIC.OSTERRATH@MILA.QUEBEC
Irina Rish IRINA.RISH@MILA.QUEBEC
Mila, Université de Montréal
Lin Dong DONGLBJ@CN.IBM.COM
IBM, Beijing, China
Chao Xue XUECHAO@CN.IBM.COM
IBM Research, Beijing, China
Junfeng Liu JUNFENG.LIU@CA.IBM.COM
Sean Wagner WAGNERSE@CA.IBM.COM
Yonggang Hu YHU@CA.IBM.COM
IBM Canada, Markham, Ontario, Canada

Editor:

Abstract

We present Oríon, a black-box optimization tool that is designed to adapt to the workflow of machine learning researchers with minimal obstruction. We propose a new version control system for experiments, which can significantly improve the organization of research projects in machine learning as well as the efficiency of hyperparameter optimization. The entire tool is built with the goals of promoting reproducibility, fair benchmarking of different machine learning models, and providing a platform for the research of black-box optimization algorithms.

Keywords: Hyperparameter Optimization, Black-Box Optimization, Experiment Version Control, Reproducibility

1. Introduction

Hyperparameter optimization is one of the most time-consuming parts of research in machine learning. Several classes of models, such as deep neural networks take days or weeks to train, making the process of hyperparameter tuning even more time-consuming (Klein et al., 2016). In spite of this, the use of automatic hyperparameter optimization tools is not widespread in the community of deep learning researchers (Bouthillier and Varoquaux, 2020). This causes a serious risk of positive bias, since research is often based on incremental improvements to state-of-the-art methods. This also contributes to the problem of reproducibility when models are highly sensitive to hyperparameter values (Islam et al., 2017; Lucic et al., 2018; Melis et al., 2018; Dodge et al., 2019).

The lack of wide-spread adoption cannot be blamed on absence of frameworks for hyperparameter optimization as they are numerous (Akiba et al., 2019; Bergstra et al., 2015; Dewancker et al., 2016; Hutter et al., 2011; Kandasamy et al., 2019; Liaw et al., 2018; Mendels and Lahav, 2018; Olson et al., 2016). We presume that the most important reason for the low adoption rate of these frameworks is the cognitive overhead incurred by using them. In an attempt to address this, we developed Oríon[1], based on a different approach centered on the following idea: **machine learning researchers must be viewed as users**, not as developers. From this perspective it follows that hyperparameter optimization should be adapted to the workflow of researchers, rather than imposed as an API to adapt their code to.

In order to make such an adaptation to the workflow, we propose a **non-intrusive** way of communicating with the user's script. We designed the tool to support user scripts written in any language or framework. It supports definition of the search space using any text based configuration files or directly using the command line. We built it to work **asynchronously** as a way to avoid the need of setting up master and workers, and to improve **resiliency**. Finally, we made it incrementally configurable for flexibility and simplicity. To further improve the research workflow, we developed a new experiment version control system for proper experiment management and boosted hyperparameter optimization. With community development in mind, we designed Oríon to be modular and support external contributions as plug-ins. Supporting contributions is an important part of our tool, as one of our main goals is to support research in the area of hyperparameter optimization.

2. Black-Box Optimization API adapted to the Research Workflow

Oríon is meant to be simple to configure and operate. It requires little adaptation from the researcher and is compatible with any programming language. We illustrate the core features with a minimal example.

Suppose the user is using the script presented on the right to train and evaluate a model with hyperparameter `x`. The user

```
import sys
from orion.client \
    import report_objective

x = float(sys.argv[2])
y = train_and_eval(x)

report_objective(y)
```

1. https://github.com/Epistimio/orion

would execute this script in command line passing a value for the hyperparameter `x`.

```
$ python script.py -x 5
```

To make this script compatible with Oríón, the only modification required is to report the objective with `report_objective`. We use Python as an example because Oríón provides a convenient helper function, but the script could be of any programming language. The only requirement is that the script saves the results in a json file following Oríón's format standard.

```
$ orion hunt --name exp python \
  script.py -x~'uniform(-5, 5)'
```

Running the script as it is would however only train and evaluate for a given value of `x`. To proceed with hyperparameter optimization, the user must call `orion hunt` and turn `-x 5` into `-x~'uniform(-5, 5)'`. The later, `uniform(-5, 5)`, is a prior which defines the search space that should be explored for a given hyperparameter. Oríón supports more than 50 distributions, their discretized versions and categorical choices.

```
$ orion hunt --name exp &
$ orion hunt --name exp &
$ orion hunt --name exp &
$ orion hunt --name exp
```

During the execution of `orion hunt`, a worker will be spawned to perform the hyperparameter optimization. The worker will connect to a database and register a new experiment with name `exp` or retrieve the corresponding experiment if it already exists. Scaling the optimizations in parallel across multiple workers is as simple as calling `orion hunt` multiple times. As illustrated in Figure 1, each worker will maintain a copy of the optimization algorithm, which will be synchronized at each updates.

This replication strategy simplifies the scaling up of parallel hyperparameter optimization. There is no need for a persistent master, and the scaling can be dynamic as the availability of resources evolves over time. This strategy also improves resilience of the whole parallel optimization process. Workers can die without having any effect for the other workers. The only single point of failure is the database itself.

3. Features

Plugins In order to easily support various use cases, many components of Oríón are made to be *pluginnable*. This means that components such as algorithms or databases can be implemented externally without requiring modifications to the core code from the researchers. A cookiecutter template[2] facilitates the creation of plugins for users.

Algorithms Oríón provides a core set of algorithms covering a wide set of trade-offs. Multi-fidelity algorithms Hyperband Li et al. (2016), ASHA Li et al. (2020) and Evolutionary Early-Stopping So et al. (2019) allow fast optimization when learning curves of optimized task are sufficiently monotonous. The Bayesian optimization algorithm TPE Bergstra et al. (2011) is a powerful candidate when hyperparameters of the search space are weakly correlated and is well suited for parallel optimization. Bayesian optimization algorithms based on Gaussian Processes on the other hand are less suited for parallel opti-

2. `https://github.com/Epistimio/cookiecutter-orion.algo`

mization but can better leverage correlated hyperparameters. As a last resort, the random search method Bergstra and Bengio (2012) can provide good results, often surprisingly close to more sophisticated algorithms.

Native implementations in Oríon include Random Search, Hyperband, ASHA, Evolutionary Early-Stopping and TPE. Two external plugins further provide wrappers to access the Bayesian optimization algorithms of scikit-optimize (orion.algo.skopt (add url to pypi)) and RoBO (orion.algo.robo (add url to pypi)). The RoBO wrapper also supports BOHAMIANN (Springenberg et al., 2016), DNGO (Snoek et al., 2015) and ABLR (Perrone et al., 2018), algorithms that can be used for more stable (BOHAMIANN) or cheaper computations (DNGO and ABLR).

Experiment Version Control Unique to Oríon, an experiment versioning system ensures coherence of code execution within one experiment, tracks changes between executions and provides a native warm-starting mechanism for the optimization algorithms.

Whenever a worker is spawned, Oríon will check whether the configuration of the experiment changed. Changes detected include those in the search space definition, algorithm configuration, command line call, user's script configuration, Oríon code version or the user's code version[3]. If such changes are detected, Oríon will *branch* the experiment, creating a new one with the new configuration, increment the version number of the new child experiment and connect it to its parent – the initial experiment. The collection of branched experiments is represented as an experiment version control tree in which users can easily track changes throughout their experimentations. Furthermore, this experiment

3. Code version is tracked only if user's script is located in a git repository.

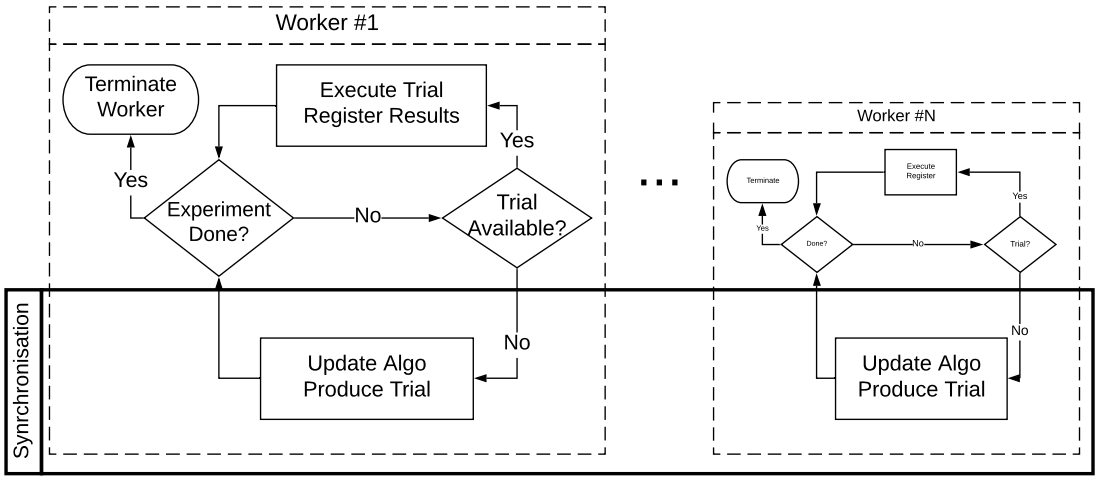

Figure 1: **Distributed optimization through replication of algorithm's state**: The flow chart illustrates the optimization process for each worker in a pool of N workers. The algorithm state is replicated across all workers, and synchronized at the time of update to produce new trials.

	Ax	HyperOpt	Katib	Nevergrad	NNI	Optuna	Ray-Tune	SMAC3	Orion
Languages supported	Py.	Py.	Any	Py. & R	Py.	Py.	Py.	Py.	Any
Disruptive	Low	Mild	Low	Low	Low	High	Mild	High	Low
Scheduling	Any	Any	K8	Any	K8 AML	Any	K8 Slurm	Part.	Any
Auto-scalability	No	Part.	Yes	Part.	Yes	Part.	Yes	Part.	Part.
Fault tolerance	No	Part.	Yes	No	Part.	Part.	Yes	No	Part.
Visualizations	High	Min	Min	Min	Good	High	Min	No*	High
Algorithms									
Bayesian Optimization	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Bandit	Yes	No	Yes	Yes*	Yes	Yes	Yes	Yes	Yes
Evolutionary	No	No	Yes	Yes*	Yes	Yes	Yes	No	Yes
Strategies for efficiency									
Conditional	No	Part.	No	No	Part.	Yes	Yes	Yes	No
Multi-fidelity	Yes	No	Yes	No	Part.	Part.	Part.	Yes	Yes
Warm-starting	Part.	Part.	No	Yes	No	Part.	Part.	Part.	Yes

Table 1: Comparison of features in open-source hyperparameter optimization libraries.

version control tree is used by Oríon to pass trials from parents down to child experiments to warm-start the optimization algorithm in the child experiments.

4. Related

An overview of the ecosystem of open-source hyperparameter optimization libraries is presented in Table 1. Oríon is one of the rare frameworks to support any programming language, to be non-intrusive and be compatible with any scheduling system. Katib George et al. (2020) offers most of these features as well but its strong dependence to Kubernetes severely hampers its usability on academic computational resources.

5. Conclusion

Thanks to the Experiment Version Control system we introduced and the features which enable the hierarchy and the modularity of algorithms, experiments and their trials, Oríon is a simple but powerful experimentation platform. Its intuitive and flexible user interface, seamless and fast integration with any research code, as well as its distributed and asynchronous approach, make Oríon an accessible and versatile tool for creating precise and organized work.

Contributors

In alphabetical order:

Guillaume Alain, Frédéric Bastien, Christopher Beckham, Arnaud Bergeron, Hadrien Bertrand, Olexa Bilaniuk, Steven Bocco, Xavier Bouthillier, Olivier Breuleux, Mirko Bronzi, François Corneau-Tremblay, Pierre Delaunay, Lin Dong, Reyhane Askari Hemmat, Peter Henderson, Yonggang Hu, Pascal Lamblin, Junfeng Liu, Michael Noukhovitch, Satya Ortiz-Gagné, Frédéric Osterrath, Irina Rish, Thomas Schweizer, Dmitriy Serdyuk, Dendi Suhubdy, Christos Tsirigotis, Sean Wagner and Chao Xue.

Acknowledgments

We are grateful to Angelos Katharopoulos, Ryan Turner, Philip Paquette, Olexa Bilaniuk, Nasim Rahaman, Mathieu Germain, Massimo Caccia, Breandan Considine, Zhouhan Lin, Dmitriy Serdyuk and Jie Fu for the insightful discussions and comments.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2623–2631, 2019.

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 System for ML*, 2018.

James Bergstra and Yoshua Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13:281–305, 2012. ISSN 1532-4435. doi: 10.1162/153244303322533223.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. *Advances in Neural Information Processing Systems (NIPS)*, pages 2546–2554, 2011. ISSN 10495258. doi: 2012arXiv1206.2944S. URL `https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf`.

James Bergstra, Daniel Yamins, and David Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123. PMLR, 2013.

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.

A. Biedenkapp, J. Marben, M. Lindauer, and F. Hutter. CAVE: Configuration assessment, visualization and evaluation. In *Proceedings of the International Conference on Learning and Intelligent Optimization (LION'18)*, 2018.

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.

Ian Dewancker, Michael McCourt, Scott Clark, Patrick Hayes, Alexandra Johnson, and George Ke. A strategy for ranking optimization methods using multiple criteria. In *Workshop on Automatic Machine Learning*, pages 11–20, 2016.

Jesse Dodge, Suchin Gururangan, Dallas Card, Roy Schwartz, and Noah A. Smith. Show your work: Improved reporting of experimental results. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2185–2194, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1224. URL https://www.aclweb.org/anthology/D19-1224.

Stefan Falkner, Aaron Klein, and Frank Hutter. Combining Hyperband and Bayesian Optimization. *Nips*, (Nips), 2017. URL http://ml.informatik.uni-freiburg.de/papers/17-BayesOpt-BOHB.pdf.

Johnu George, Ce Gao, Richard Liu, Hou Gang Liu, Yuan Tang, Ramdoot Pydipaty, and Amit Kumar Saha. A scalable and cloud-native hyperparameter tuning system. *arXiv preprint arXiv:2006.02085*, 2020.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International conference on learning and intelligent optimization*, pages 507–523. Springer, 2011.

Riashat Islam, Peter Henderson, Maziar Gomrokchi, and Doina Precup. Reproducibility of benchmarked deep reinforcement learning tasks for continuous control. *arXiv preprint arXiv:1708.04133*, 2017.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population Based Training of Neural Networks. 2017. URL http://arxiv.org/abs/1711.09846.

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. *arXiv preprint arXiv:1903.06694*, 2019.

A. Klein, S. Falkner, N. Mansur, and F. Hutter. Robo: A flexible and robust bayesian optimization framework in python. In *NIPS 2017 Bayesian Optimization Workshop*, December 2017.

Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *CoRR*, abs/1605.07079, 2016.

Liam Li, Kevin Jamieson, Afshin Rostamizadeh, Ekaterina Gonina, Jonathan Bentzur, Moritz Hardt, Benjamin Recht, and Ameet Talwalkar. A system for massively parallel hyperparameter tuning. In I. Dhillon, D. Papailiopoulos, and V. Sze, editors, *Proceedings of Machine Learning and Systems*, volume 2, pages 230–246, 2020. URL `https://proceedings.mlsys.org/paper/2020/file/f4b9ec30ad9f68f89b29639786cb62ef-Paper.pdf`.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. 2016.

Zhao Lucis Li, Chieh-Jan Mike Liang, Wenjia He, Lianjie Zhu, Wenjun Dai, Jin Jiang, and Guangzhong Sun. Metis: Robustly tuning tail latencies of cloud systems. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 981–992, 2018.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter. Boah: A tool suite for multi-fidelity bayesian optimization & analysis of hyperparameters. *arXiv:1908.06756 [cs.LG]*.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, Stefan Falkner, André Biedenkapp, and Frank Hutter. Smac v3: Algorithm configuration in python. `https://github.com/automl/SMAC3`, 2017.

Mario Lucic, Karol Kurach, Marcin Michalski, Sylvain Gelly, and Olivier Bousquet. Are GANs Created Equal? A Large-Scale Study. *arXiv:1711.10337 [cs, stat]*, November 2018.

Gábor Melis, Chris Dyer, and Phil Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations*, 2018.

Gideon Mendels and Nimrod Lahav. Comet.ml - supercharging machine learning, May 2018. URL `https://www.comet.ml`.

Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. Springer International Publishing, 2016. ISBN 978-3-319-31204-0. doi: 10.1007/978-3-319-31204-0_9. URL `http://dx.doi.org/10.1007/978-3-319-31204-0_9`.

Valerio Perrone, Rodolphe Jenatton, Matthias Seeger, and Cédric Archambeau. Scalable hyperparameter transfer learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 6846–6856, 2018.

J. Rapin and O. Teytaud. Nevergrad - A gradient-free optimization platform. `https://GitHub.com/FacebookResearch/Nevergrad`, 2018.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

David So, Quoc Le, and Chen Liang. The evolved transformer. In *International Conference on Machine Learning*, pages 5877–5886. PMLR, 2019.

Jost Tobias Springenberg, Aaron Klein, Stefan Falkner, and Frank Hutter. Bayesian optimization with robust bayesian neural networks. *Advances in neural information processing systems*, 29:4134–4142, 2016.

Appendix A. Criteria for comparison of frameworks

A.1 Usability

These criteria attests for the ease of use of the libraries. How quickly the user can get up to speed, how easily it can debug and inspect processes.

A.1.1 Languages supported

The programming languages supported are languages used by the user to execute the black-box process. It may be different than the language used to implement the library itself.

A.1.2 Disruptive

The library is considered disruptive for the researcher's workflow if it requires substantial modifications. Libraries that offer flexible interfaces are less disruptive while libraries that requires very specific implementations are highly disruptive.

Low The code requires only 1-5 lines of modifications and does not affect the control flow of the user's program.

Mild The code may require more than 5 lines of code or affect the control flow of the user's program.

High The code requires more than 5 lines of code and affect the control flow of the user's program.

A.1.3 Scheduling

Some libraries are limited to specific scheduling systems (ex: through Kubernetes). This criteria evaluates whether a library supports only a few schedulers or if they are compatible with any.

A.1.4 Auto-scalability

Libraries integrated with scheduling systems may support auto-scaling, that is, the capacity of dynamically allocating computational resources and spawning workers according to the needs of the hyperparameter optimization process.

A.1.5 Fault tolerance

Hyperparameter optimization at scale may fail on several fronts. Fault tolerance is an important feature to streamline the optimization. Faults considered are errors during the black box optimization, crash of the optimization algorithm process or of the worker.

A.1.6 Visualizations

Visualisation is crucial tool to allow inspection of the optimization process and interpretation of the results.

No No vizualisations.

Min Minimal plots: Regret plot

Good Coverage of a good set of vizualizations: Regret, Parallel Coordinate plots

High Coverage of a large set of vizualizations: Regret, Parallel Coordinate plots, Optimization space, and more.

A.2 Algorithms

These high level criteria attest for the variety of algorithms supported by the libraries.

A.2.1 Bayesian Optimization

We consider any type of Bayesian optimization, from classical Gaussian process based Bayesian Optimization to Tree Structured Parzen Estimator (TPE) Bergstra et al. (2011).

A.2.2 Bandit

Arm-based optimization algorithms fall in the *Bandit* category. These include Hyperband Li et al. (2016) and ASHA Li et al. (2020).

A.2.3 Evolutionary

We include in the *evolutionary* category any algorithm using mutations. This, among others, includes Population Based Training (PBT) Jaderberg et al. (2017) and Covariance matrix adaptation evolution strategy (CMA-ES) Hansen and Ostermeier (2001).

A.3 Strategies for efficiency

Hyperparameter optimization is computationally expensive. We include in this comparison three features that improves its efficiency.

A.3.1 Conditional

Conditional search space helps reducing the explorable space and thus can speed up hyperparameter optimization dramatically. A library may support one or many of the points below. We report *Yes* in Table 1 if all points are supported, *Part.* otherwise.

- Hierarchical choices. Ex: Randomly selecting one of different group of choices and then randomly selecting a choice within this group.
- Conditional choices. Ex: The group of choices to sample from is conditional to the value of another hyperparameters.
- Conditionally constrained space. Ex: The possible values is constrained based on values of other hyperparameters. This may be applied to any type of prior, not only choices.

A.3.2 Multi-fidelity

Multi-fidelity speeds up hyperparameter optimization by looking at low fidelity first (ex: objective value after few epochs of training) to guide selection of hyperparameters to try at higher fidelity. For instance, libraries that includes Hyperband Li et al. (2016) or ASHA Li et al. (2020) supports multi-fidelity.

A.3.3 Warm-starting

Algorithms like Bayesian optimization can converge faster if provided with prior data, a procedure known as warm-starting.

No No warm-start possible.

Part. Requires the user to manually feed the algorithm with data.

Yes Supports warm-starting through automated transfert of data from one experiment to another.

Appendix B. Evaluation of frameworks

B.1 Ax (Bakshy et al., 2018)

B.1.1 Usability

Languages supported Only supports python.

Disruptive (Low)
The loop API allows defining an optimization loop with a single function call, but is limited to sequential optimization. The service API requires few additional lines to define the client, retrieve parameters and send results. The service API can easily fit in most research workflows.

Scheduling There is no scheduling system in Ax. It must be integrated with the service of the developer API.

Auto-scalability There is no scheduling system and thus no auto-scalability capability.

Fault tolerance No automatic handling of failing trials is provided in the different APIs.

Visualizations (High)
Ax offers a large variety of plots. Among many, there are contour plots of the search space, feature importance plots, marginal effects (equivalent to partial dependencies), pareto plots for multi-objective optimization, slice plots to represent 1-d views of the search space, regret plot with support for averages across runs and bar plots of optimization times.

B.1.2 Algorithms

Bayesian Optimization Wrapping on top of BoTorch, Ax provides flexible and efficient solutions for Bayesian Optimizations.

Bandit Ax allows a factorial design of the experiments which enables bandit optimization. The use of the factorial design requires however significantly more work from the researchers than out-of-the-box implementations of algorithms such as Hyperband Li et al. (2016).

Evolutionary There are no Evolutionary algorithms in Ax.

B.1.3 Strategies for efficiency

Conditional (No)
There is no support for conditional search spaces.

Multi-fidelity (Yes)
Multi-fidelity can be achieved with the multi-factorial design of the experiments.

Warm-starting (Part.)
The Bayesian optimization algorithms of BoTorch can be manually fitted on prior data by the user. There is no system that automates warm-starting.

B.2 HyperOpt (Bergstra et al., 2013)

B.2.1 Usability

Languages supported Only supports python.

Disruptive (Mild)
The fmin function executes sequential optimization with a single function call. It can be turned into a master process if using the MongoDB trials. In this case workers must be spawned separately. The arguments passed to the function are not unpacked and requires additional tinkering from the user to support calls from fmin.

Scheduling (Any)
There is no scheduling system in HyperOpt, but it can be interfaced with a Spark cluster using the Spark Trials.

Auto-scalability (Part.)
Only possible through Spark's dynamic resource allocation when using spark trials.

Fault tolerance (Part.)
Only when using spark trials.

Visualizations (Min)
The repository contains 4 undocumented functions to plot time-series of objectives, histogram of objectives, per-dimension scatter plots of dimension values and objectives, and a line plot of attachment data saved in trials.

B.2.2 Algorithms

Bayesian Optimization Hyperopt is the reference for the original implementation of the Tree Structured Parzen Estimator (TPE) Bergstra et al. (2011).

Bandit None

Evolutionary None

B.2.3 Strategies for efficiency

Conditional (Part.)
Hyperopt supports hierarchical choices.

Multi-fidelity None

Warm-starting (Part.)
The user may warm-start the TPE by manually inserting data before starting the optimization.

B.3 Katib (George et al., 2020)

B.3.1 Usability

Languages supported Any languages are supported.

Disruptive (High)
On the user's code side, Katib is fairly non-intrusive supporting hyperparameter definition through commandline arguments or environment variables. The setup required to run it with kubeflow is however very disruptive for any researcher not already using it.

Scheduling Natively supported through kubeflow.

Auto-scalability Natively supported through kubeflow.

Fault tolerance Supported through kubeflow.

Visualizations (Min)
Katib provides a dashboard including a parallel coordinates plot and a table of trial results.

B.3.2 Algorithms

Bayesian Optimization Katib interfaces with hyperopt, which contains original python implementation of TPE Bergstra et al. (2011), GOptuna, which contains native re-implementation of TPE in Go language, skopt (bayesian opt), chocolate which contains a native implementation of bayesian optimization based on scikit-learn's gaussian processes.

Bandit Katib includes a native implementation of Hyperband Li et al. (2016).

Evolutionary Katib includes CMA-ES.

B.3.3 Strategies for efficiency

Conditional We have found no support for any form of conditional search spaces in Katib.

Multi-fidelity Multi-fidelity is supported through Hyperband Li et al. (2016).

Warm-starting We have found no support for warm-starting in Katib.

B.4 Nevergrad (Rapin and Teytaud, 2018)

B.4.1 Usability

Languages supported Python and R

Disruptive (Low)
Setting up experiment with Nevergrad requires only a few lines of code. It is also possible to adapt to most user workflows using the ask/tell interface.

Scheduling There is no scheduling system in Nevergrad. It must be integrated with the ask/tell interface.

Auto-scalability There is no scheduling system, but a `batch mode` is available to help parallelize workers locally using `concurrent.futures.ThreadPoolExecutor`.

Fault tolerance We have found no mechanisms for fault tolerance.

Visualizations (Min)
Nevergrad supports plotting of the regret curve along with a ranking frequency matrix called *fight plots* in the library.

B.4.2 Algorithms

Bayesian Optimization An optimizer wrapping the library BayesianOptimization[4] provides support for Bayesian Optimization.

Bandit The only algorithm supported is NoisyBandit.

Evolutionary Among the libraries considered, Nevergrad is by far the one containing the largest number of evolutionary algorithm implementations. It is also by far the one containing the most gradient-free optimization methods.

B.4.3 Strategies for efficiency

Conditional We have found no support for conditional dimensions in Nevergrad.

Multi-fidelity We have found no support for multi-fidelity in Nevergrad.

Warm-starting Nevergrad supports a form of warm-starting through *chaining*. The *chaining* of algorithms consists in defining a sequence of algorithms that will be used one after the other when given budgets of trials are reached. The trials observed by a given algorithm is passed to the next one.

B.5 NNI

B.5.1 Usability

Languages supported Python

Disruptive (Low)
NNI can easily be integrated in most code using `nni.get_next_parameter()` and `nni.report_final_result`

4. github.com/fmfn/BayesianOptimizer

Scheduling NNI provides support for many schedulers. It supports multiple services based on Kubernetes (OpenPAI, Kubeflow, AKS) as well as Azure Machine Learning and ssh-based scheduling.

Auto-scalability Auto-scalability can be achieved with most schedulers supported by NNI.

Fault tolerance We could not find fault tolerance options in NNI itself. The schedulers can offer some form of fault tolerance however.

Visualizations The dashboard in NNI provides a regret curve and parallel coordinates plots.

B.5.2 Algorithms

Bayesian Optimization NNI provides wrappers for BOHB Falkner et al. (2017), TPE Bergstra et al. (2011), SMAC, a native Metis Tuner Li et al. (2018) and a Bayesian Optimizer based on sklearn.

Bandit NNI provides Hyperband Li et al. (2016).

Evolutionary NNI provides the Population Based Training algorithm Jaderberg et al. (2017).

B.5.3 Strategies for efficiency

Conditional NNI supports hierarchical choices with the `choice` parameter type.

Multi-fidelity Multi-fidelity is supported through Hyperband Li et al. (2016). It is implemented as a combination of `Tuner` and `Assessor` which makes it only support time based fidelities.

Warm-starting We have found no ways to warm-start algorithms with NNI.

B.6 Optuna (Akiba et al., 2019)

B.6.1 Usability

Languages supported Python

Disruptive (High)
The *pythonic search space* of Optuna allows for more flexibility but makes it more disruptive to adapt existing code.

Scheduling Optuna is compatible with any scheduling system.

Auto-scalability Optuna does not support auto-scalability natively but its integration with Dask and Ray makes it possible to scale executions of Optuna.

Fault tolerance Failing trials will cause the optimizer to end unless the user defines special types of exceptions that must be ignored.

Visualizations (High)

Optuna offers a rich set of visualizations: Regret plot, parallel coordinate plots, hyperparameter importance, partial dependency plot (called contour plots in Optuna), pareto front for multi-objective experiments, and more.

B.6.2 Algorithms

Bayesian Optimization Optuna provides a native implementation of TPE Bergstra et al. (2011) including support for multi-dimensional optimization (through `sample_relative()`).

Bandit Optuna provides an implementation of Hyperband Li et al. (2016) as a 'pruner'.

Evolutionary Optuna provides a wrapper for the library `cmaes`[5].

B.6.3 Strategies for efficiency

Conditional Optuna has one of the best support for conditional distributions thanks to its *pythonic search space*.

Multi-fidelity Optuna provides an implementation of Hyperband Li et al. (2016) as a `Pruner`. It is more constraining than a general implementation for which the fidelity is not assumed to be related to training time, but can still be considered as multi-fidelity.

Warm-starting Optuna supports fixing hyperparameters to warm-start algorithms in smaller search space using the `PartialFixedSampler`.

B.7 Ray-Tune (Liaw et al., 2018)

B.7.1 Usability

Languages supported Python

Disruptive (Mild)

Most features of Ray-Tune can be used with few lines of codes. The user is required however to define a function that will be optimized, thus constraining the supported workflows.

Scheduling Scheduling is done through Ray, which supports Kubernetes for the cloud and Slurm for HPC.

Auto-scalability Ray allows auto-scalability through the cluster managers and ray serve.

Fault tolerance Ray-Tune supports rescheduling of crashing trials.

Visualizations Ray-Tune natively generates TensorBoard files which can be used to visualize parallel coordinates plots.

5. github.com/CyberAgent/cmaes

B.7.2 Algorithms

Bayesian Optimization Through extensive wrappers, Ray-Tune supports multiple variants of Bayesian Optimization: `AxSearch`, `DragonflySearch`, `SkoptSearch`, `BayesOptSearch`, `TuneBOHB`, `OptunaSearch` and `NevergradSearch`.

Bandit Ray-Tune have native implementations of both Hyperband Li et al. (2016) and ASHA Li et al. (2020). It also supports BOHB Falkner et al. (2017) through a wrapper.

Evolutionary Ray-Tune have a native implementation of Population Based Training (PBT) Jaderberg et al. (2017). It also supports other evolutionary algorithms through the wrappers `NevergradSearch` and `OptunaSearch`.

B.7.3 Strategies for efficiency

Conditional Ray-Tune supports very flexible conditional search spaces with the method `tune.sample_from()` that can access other hyperparameter values and adjust accordingly.

Multi-fidelity Ray-Tune supports multi-fidelity with algorithms Hyperband, ASHA and BOHB. All these algorithms are implemented in the form of schedulers however which limits them to time-based fidelities.

Warm-starting Users can manually pass a list of trials to try initially if no objectives given, or warm-start if objectives are given using argument `points_to_evaluate`.

B.8 SMAC3 (Lindauer et al., 2017)

B.8.1 Usability

Languages supported Python

Disruptive (High)

SMAC3 only supports optimizing functions that complies to SMAC3 interfaces, which constraints the possible workflows for users. The search space used by SMAC3, `ConfigSpace`, is one of the most powerful, but it is more verbose, more complex and requires more code than most other libraries.

There exists an simpler interface `fmin_smac` for the optimization of functions with simple search spaces.

Scheduling SMAC3 relies on file-system to share information across workers by default. It can be used with any scheduler. SMAC3 also provides a bridge to Dask.

Auto-scalability SMAC3 can support auto-scalability when used with Dask.

Fault tolerance We found no mechanisms for fault tolerance.

Visualizations SMAC3 provides no visualization tools natively but is compatible with the tool CAVE Biedenkapp et al. (2018); Lindauer et al. developed by the same research group.

B.8.2 Algorithms

Bayesian Optimization SMAC3 is a reference for high quality Bayesian Optimization algorithms.

Bandit SMAC3 supports Hyperband Li et al. (2016) and BOHB Falkner et al. (2017).

Evolutionary None.

B.8.3 Strategies for efficiency

Conditional The library `ConfigSpace` used by SMAC3 fully supports conditional search spaces.

Multi-fidelity SMAC3 supports Hyperband Li et al. (2016) and BOHB Falkner et al. (2017).

Warm-starting The user can manually pass previous runs to the optimizer with `runhistory` to warm-start it.

B.9 Oríon

B.9.1 Usability

Languages supported Any.

Disruptive Assuming the user script supports hyperparameter definitions using arguments, Oríon only requires adding one line of code. Otherwise, the python API with `suggest`/`observe` can support most workflows.

Scheduling Any.

Auto-scalability (Part.)
Oríon can support auto-scalability when used with Dask.

Fault tolerance (Part.)
The user can define a number of broken trials that may be tolerated, but broken trials are not re-executed automatically. If a worker crash it is not automatically respawned.

Visualizations Oríon provides regret, local parameter importance, parallel coordinate plots and partial dependency plots.

B.9.2 Algorithms

Bayesian Optimization A native implementation of Tree-Structure Parzen Estimator Bergstra et al. (2011) is included in Oríon. Oríon provide wrappers for scikit-optimize and RoBO Klein et al. (2017). This gives access to Bayesian Optimization with Gaussian Processes and Random Forests, BOHAMIANN (Springenberg et al., 2016), DNGO Snoek et al. (2015) and ABLR (Perrone et al., 2018).

Bandit Oríon includes native implementations of Hyperband Li et al. (2016) and ASHA Li et al. (2020).

Evolutionary The EvolutionaryES algorithm So et al. (2019) is implemented in Oríon.

B.9.3 Strategies for efficiency

Conditional None.

Multi-fidelity Hyperband and ASHA implementations in Oríon supports any type of fidelity.

Warm-starting The Experiment Version Control provides seamless warm-starting under the hood.