Sherpa: Hyperparameter Optimization for Machine Learning Models

Lars Hertel
Department of Statistics
University of California, Irvine
lhertel@uci.edu

Julian Collado
Department of Computer Science
University of California, Irvine
colladou@uci.edu

Peter Sadowski
University of Hawaii
peter.sadowski@hawaii.edu

Pierre Baldi
Department of Computer Science
University of California, Irvine
pfbaldi@ics.uci.edu

Abstract

Sherpa is a free open-source hyperparameter optimization library for machine learning models. It is designed for problems with computationally expensive iterative function evaluations, such as the hyperparameter tuning of deep neural networks. With Sherpa, scientists can quickly optimize hyperparameters using a variety of powerful and interchangeable algorithms. Additionally, the framework makes it easy to implement custom algorithms. Sherpa can be run on either a single machine or a cluster via a grid scheduler with minimal configuration. Finally, an interactive dashboard enables users to view the progress of models as they are trained, cancel trials, and explore which hyperparameter combinations are working best. Sherpa empowers machine learning researchers by automating the tedious aspects of model tuning and providing an extensible framework for developing automated hyperparameter-tuning strategies. Its source code and documentation are available at `https://github.com/LarsHH/sherpa` and `https://parameter-sherpa.readthedocs.io/`, respectively. A demo can be found at `https://youtu.be/L95sasMLgP4`.

1 Existing Hyperparameter Optimization Libraries

Hyperparameter optimization algorithms for machine learning models have previously been implemented in software packages such as Spearmint [15], HyperOpt [2], Auto-Weka 2.0 [9], and Google Vizier [5] among others.

Spearmint is a Python library based on Bayesian optimization using a Gaussian process. Hyperparameter exploration values are specified using the markup language YAML and run on a grid via SGE and MongoDB. Overall, it combines Bayesian optimization with the ability for distributed training.

HyperOpt is a hyperparameter optimization framework that uses MongoDB to allow parallel computation. The user manually starts workers which receive tasks from the HyperOpt instance. It offers the use of Random Search and Bayesian optimization based on a Tree of Parzen Estimators.

Auto-WEKA 2.0 implements the SMAC [6] algorithm for automatic model selection and hyperparameter optimization within the WEKA machine learning framework. It provides a graphical user interface and supports parallel runs on a single machine. It is meant to be accessible for novice users and specifically targets the problem of choosing a model. Auto-WEKA is related to Auto-Sklearn [4] and Auto-Net [11] which specifically focus on tuning Scikit-Learn models and fully-connected

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Table 1: Comparison to Existing Libraries

	Spearmint	Auto-WEKA	HyperOpt	Google Vizier	Sherpa
Early Stopping	No	No	No	Yes	Yes
Dashboard/GUI	Yes	Yes	No	Yes	Yes
Distributed	Yes	No	Yes	Yes	Yes
Open Source	Yes	Yes	Yes	No	Yes
# of Algorithms	2	1	2	3	5

neural networks in Lasagne, respectively. Auto-WEKA, Auto-Sklearn, and Auto-Net focus on an end-to-end automatic approach. This makes it easy for novice users, but restricts the user to the respective machine learning library and the models it implements. In contrast our work aims to give the user more flexibility over library, model and hyper-parameter optimization algorithm selection.

Google Vizier is a service provided by Google for its cloud machine learning platform. It incorporates recent innovation in Bayesian optimization such as transfer learning and provides visualizations via a dashboard. Google Vizier provides many key features of a current hyperparameter optimization tool to Google Cloud users and Google engineers, but is not available in an open source version. A similar situation occurs with other cloud based platforms like Microsoft Azure Hyperparameter Tuning [1] and Amazon SageMaker's Hyperparameter Optimization [2].

2 Need for a new library

The field of machine learning has experienced massive growth over recent years. Access to open source machine learning libraries such as Scikit-Learn [14], Keras [3], Tensorflow [1], PyTorch [13], and Caffe [8] allowed research in machine learning to be widely reproduced by the community making it easy for practitioners to apply state of the art methods to real world problems. The field of hyperparameter optimization for machine learning has also seen many innovations recently such as Hyperband [10], Population Based Training [7], Neural Architecture Search [17], and innovation in Bayesian optimization such as [16]. While the basic implementation of some of these algorithms can be trivial, evaluating trials in a distributed fashion and keeping track of results becomes cumbersome which makes it difficult for users to apply these algorithms to real problems. In short, Sherpa aims to curate implementations of these algorithms while providing infrastructure to run these in a distributed way. The aim is for the platform to be scalable from usage on a laptop to a computation grid.

3 Key Features

3.1 Choice of Algorithms

A key motivation for Sherpa is to provide implementations of recent hyperparameter tuning algorithms to users while making it easy to add new algorithms. The currently implemented algorithms are:

- Random Search
- Grid Search
- Local Search: increasing or decreasing one hyperparameter at a time
- Bayesian Optimization using a Gaussian Process and Expected Improvement Acquisition function
- Population Based Training (PBT)[7].

The documentation provides tutorials for using Bayesian Optimization, PBT, and Local Search and how to implement a new hyperparameter search algorithm.

[1] `https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/tune-model-hyperparameters`

[2] `https://aws.amazon.com/blogs/aws/sagemaker-automatic-model-tuning/`

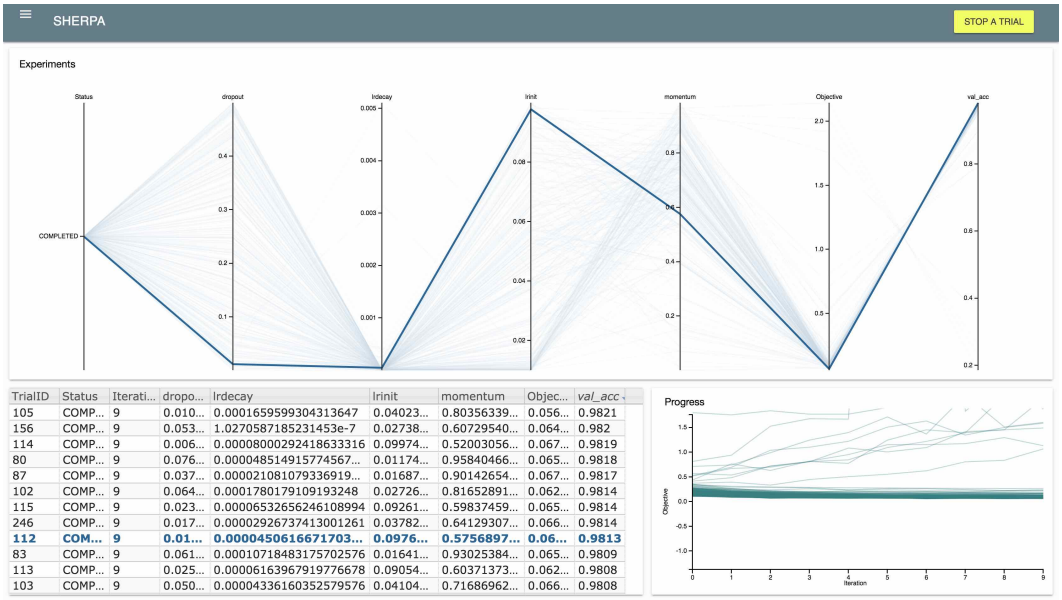
Sherpa also allows to implement stopping rules, decision rules which allow stopping of underperforming trials. A trial is the training of a model with a specific set of hyperparameters. Sherpa currently implements a Median-Stopping-Rule [5] which automatically stops any trial with lower performance than the median of the finished trials at the same epoch. Additional information can be found in the documentation.

3.2 Scalable Computation via Plug-In Schedulers

Sherpa is meant to be flexible around the user's computational resources. It can simply be run in a single Python session such as a Jupyter Notebook, or in parallel using two scripts. The first Python script then runs the Sherpa optimization and uses the Sherpa scheduler to submit jobs. A second Python script implements the trial evaluation. A database is automatically run in the background to receive metrics from trials. Sherpa submits the desired number of parallel trials and submits a new one whenever a previous trial finishes. Schedulers can execute jobs locally via subprocesses, or remotely via a grid engine such as SGE [12]. This gives the user an easy way to coordinate and run many parallel trials and it is easy to implement new schedulers.

3.3 Visualization Dashboard

Sherpa provides an interactive web-based dashboard to keep the user informed about the progress of the hyperparameter optimization. It has a table of all completed trials including their best performance and hyperparameter configuration. A line chart shows the sequence of objective values for each trial against its training iterations. Finally, a parallel coordinates plot allows the user to explore completed trials. This is very useful for noticing trends among useful configurations allowing faster and more successful hyperparameter explorations.



TrialID	Status	Iterati...	dropo...	lrdecay	lrinit	momentum	Objec...	val_acc
105	COMP...	9	0.010...	0.0001659599304313647	0.04023...	0.80356339...	0.056...	0.9821
156	COMP...	9	0.053...	1.0270587185231453e-7	0.02738...	0.60729540...	0.064...	0.982
114	COMP...	9	0.006...	0.0000800029241863331 6	0.09974...	0.52003056...	0.067...	0.9819
80	COMP...	9	0.076...	0.0000485149157745 67...	0.01174...	0.95840466...	0.065...	0.9818
87	COMP...	9	0.037...	0.0000210810793369 19...	0.01687...	0.90142654...	0.067...	0.9817
102	COMP...	9	0.064...	0.0001780179109193248	0.02726...	0.81652891...	0.062...	0.9814
115	COMP...	9	0.023...	0.0000653265624610899 4	0.09261...	0.59837459...	0.065...	0.9814
246	COMP...	9	0.017...	0.0000292673741300126 1	0.03782...	0.64129307...	0.066...	0.9814
112	**COM...**	**9**	**0.01...**	**0.0000450616671703...**	**0.0976...**	**0.5756897...**	**0.06...**	**0.9813**
83	COMP...	9	0.061...	0.0001071848317570257 6	0.01641...	0.93025384...	0.065...	0.9809
113	COMP...	9	0.025...	0.0000616396791977667 8	0.09054...	0.60371373...	0.062...	0.9808
103	COMP...	9	0.050...	0.0000433616035257957 6	0.04104...	0.71686962...	0.066...	0.9808

Figure 1: The dashboard provides a parallel coordinates plot (top) and a table of finished trials (bottom left). Trials in progress are shown via a progress line-plot (bottom right).

Furthermore, the dashboard provides an interface to interact with the optimization while it is running. A stopping button allows the user to stop trials during their training. If the user notices a trial performing badly via the line chart, she can decide to stop it and free up the resource to train a new trial.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] James Bergstra, Dan Yamins, and David D Cox. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in Science Conference*, pages 13–20. Citeseer, 2013.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

[5] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM, 2017.

[6] 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.

[7] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

[8] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[9] Lars Kotthoff, Chris Thornton, Holger H Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research*, 18(1):826–830, 2017.

[10] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv preprint arXiv:1603.06560*, 2016.

[11] Hector Mendoza, Aaron Klein, Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. Towards automatically-tuned neural networks. In *Workshop on Automatic Machine Learning*, pages 58–65, 2016.

[12] W. Gentzsch (Sun Microsystems). Sun grid engine: Towards creating a compute power grid. In *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, CCGRID '01, pages 35–, Washington, DC, USA, 2001. IEEE Computer Society.

[13] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, B. Michel, V. and Thirion, O. Grisel, M. Blondel, R. Prettenhofer, P. and Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[15] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.

[16] Jian Wu, Matthias Poloczek, Andrew G Wilson, and Peter Frazier. Bayesian optimization with gradients. In *Advances in Neural Information Processing Systems*, pages 5273–5284, 2017.

[17] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.