Hyperparameter Optimization with SHERPA

Lars Hertel, Julian Collado, Peter Sadowski, Pierre Baldi

University of California Irvine, University of Hawai‘i at Mānoa

December 7th, 2018

Need for a New Library

- Hyperparameter optimization is critical in machine learning.
- A variety of powerful algorithms have been introduced:
    - Bayesian Optimization (Multi-task BO, FABOLAS, Freeze-Thaw)
    - Bandit based methods (Hyperband, Successive Halving)
    - Evolutionary type methods (Population Based Training)
    - Neural Architecture Search (NAS, Efficient NAS, NAO)

No single algorithm is optimal in all settings, or in all stages of development; model development is a process that typically requires exploration followed by fine-tuning.

Need for a New Library

SHERPA

Enables researchers to experiment, visualize, and scale quickly.

	Spearmint	**Auto-WEKA**	**HyperOpt**	**GoogleVizier**	**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

Quickstart

To run Sherpa on a single machine, simply import Sherpa and define the parameters to be optimized, the algorithm, and how to train the model using those parameters. A pseudocode example for Keras:

```
import sherpa
study = sherpa.Study(params, algorithm)
for trial in study:
    model = define_model(trial)
    clbk  = study.keras_callback(trial)
    model.fit(X, Y, callbacks=[clbk])
    study.finalize(trial=trial)
```

Diversity of Algorithms

Optimizing hyperparameters is a process. Use one algorithm for exploration, another for fine-tuning, and yet another to satisfy those reviewers. Sherpa currently implements five core algorithms:

- Random Search
- Grid Search
- Local Search — greedy hill-climbing, one direction at a time.
- Bayesian Optimization using a Gaussian Process and Expected Improvement Acquisition function.
- Population Based Training (PBT)[2].

Custom Algorithms

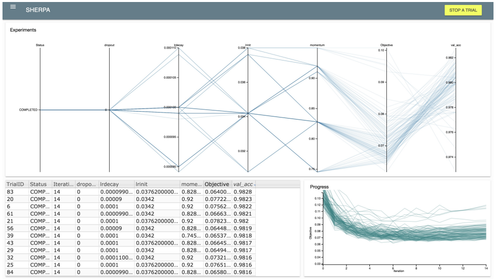
Creating custom algorithms is easy. These simply need to specify which hyperparameters to evaluate next based on the results of previous trials, and may take advantage of more information than just the final loss value, such as the entire loss trajectory of each trial or other metrics. They may also choose to start from a partially-trained model, as in the Population Based Training algorithm.

```
class CustomAlgorithm(Algorithm):
    def get_suggestion(params, results):
        ...
        return next_setting
```

Scaling Up with a Cluster

Sherpa can automatically run parallel evaluations on a cluster using a job scheduler such as SGE. Simply provide a Python script that takes a set of hyperparameters as arguments and performs a single trial evaluation. A database collects the partial results in real-time, and the hyperparameter optimization algorithm decides what to do next.

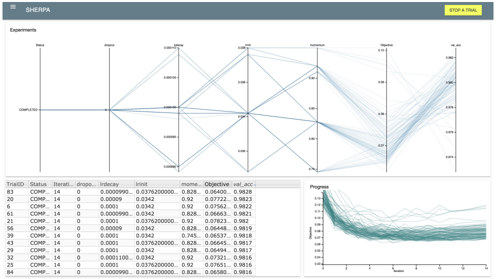
Visualization Dashboard



TrialID	Status	Iterati...	dropo...	lrdecay	lrinit	mome...	Objective	*val_acc*
83	COMP...	14	0	0.0000990...	0.0376200000...	0.828...	0.06400...	0.9828
20	COMP...	14	0	0.00009	0.0342	0.92	0.07722...	0.9823
6	COMP...	14	0	0.0001	0.0342	0.92	0.07562...	0.9822
61	COMP...	14	0	0.0000990...	0.0342	0.828...	0.06663...	0.9821
21	COMP...	14	0	0.0001	0.0376200000...	0.92	0.07823...	0.982
56	COMP...	14	0	0.00009	0.0342	0.828...	0.06448...	0.9819
39	COMP...	14	0	0.0001	0.0342	0.745...	0.06537...	0.9818
43	COMP...	14	0	0.0001	0.0376200000...	0.828...	0.06645...	0.9817
29	COMP...	14	0	0.0001	0.0342	0.828...	0.06494...	0.9817
32	COMP...	14	0	0.0001100...	0.0342	0.92	0.07321...	0.9816
25	COMP...	14	0	0.0001	0.0376200000...	0.92	0.07651...	0.9816
84	COMP...	14	0	0.0000990...	0.0376200000...	0.828...	0.06580...	0.9816

- *Parallel Coordinates:* Axes of hyperparameters and metrics.
- *Table:* Details of completed trials.
- *"Stop Trial" Button:* Stop a particular trial early.
- *Line Chart:* Trajectories of objective e.g. validation loss.

Visualization Dashboard



TrialID	Status	Iterati...	dropo...	lrdecay	lrinit	mome...	Objective	*val_acc*
83	COMP...	14	0	0.0000990...	0.0376200000...	0.828...	0.06400...	0.9828
20	COMP...	14	0	0.00009	0.0342	0.92	0.07722...	0.9823
6	COMP...	14	0	0.0001	0.0342	0.92	0.07562...	0.9822
61	COMP...	14	0	0.0000990...	0.0342	0.828...	0.06663...	0.9821
21	COMP...	14	0	0.0001	0.0376200000...	0.92	0.07823...	0.982
56	COMP...	14	0	0.00009	0.0342	0.828...	0.06448...	0.9819
39	COMP...	14	0	0.0001	0.0342	0.745...	0.06537...	0.9818
43	COMP...	14	0	0.0001	0.0376200000...	0.828...	0.06645...	0.9817
29	COMP...	14	0	0.0001	0.0342	0.828...	0.06494...	0.9817
32	COMP...	14	0	0.0001100...	0.0342	0.92	0.07321...	0.9816
25	COMP...	14	0	0.0001	0.0376200000...	0.92	0.07651...	0.9816
84	COMP...	14	0	0.0000990...	0.0376200000...	0.828...	0.06580...	0.9816

Real-time monitoring:

- Is training unstable for any HP settings?
- Do some HPs have little impact?
- Are HP ranges appropriate?

Analysis:

- How well have we explored the HP space?
- Do best HPs differ from what was expected?
- Is there a consistent pattern among the best HPs?

Recommendations

General Strategy:

- Start by picking a lot of HPs to tune with wide ranges. Then iteratively narrow down to the important HPs and appropriate ranges.

Recommendations

Grid Search:

- Useful when trying to understand the effect of one or two hyperparameters. Don't use with more hyperparameters than that. Don't use for a "global" search.

Recommendations

Random Search:

- Great for getting a full picture of the effect of all involved hyperparameters. Make sure not to discretize continuous variables or you're throwing away useful information.

Recommendations

Bayesian Optimization:

- Go to when one just wants to run one global HP optimization. Especially when model training is fast and number of hyperparameters is not too big this is optimal. More efficient than Random Search, but results will be biased.

Recommendations

Local Search:

- Use this to explore tweaks to a baseline. Does not require as many trials as Grid Search or Random Search, but has no guarantees to find global minimum.

Recommendations

Population Based Training:

- Unique in that it can find schedules for training hyperparameters (optimization HPs, regularization HPs). Great for learning rate, batch size, or momentum. Since it may be difficult to recreate schedule might want to use this last.

Summary

- Sherpa is an open-source hyperparameter optimization library for machine learning.
- Optimize your model using a variety of powerful and interchangeable algorithms.
- Write custom algorithms.
- Run on a laptop or a cluster.
- Visualize progress in an interactive dashboard.

Where to find SHERPA

```
pip install parameter-sherpa
```

https://github.com/LarsHH/sherpa

https://parameter-sherpa.rtfd.io

References

- Li et al., Hyperband: A novel bandit-based approach to hyperparameter optimization, JMLR 2018
- Jaderberg et al., Population Based Training of Neural Networks, arXiv 2017
- Swersky et al., Freeze-thaw Bayesian optimization, arXiv 2014
- Zoph et al., Neural architecture search with reinforcement learning, arXiv 2016
- Wu et al., Bayesian optimization with gradients, NIPS 2017