Tool Comparison: Optuna vs Hyperopt

Audrey Bertin, Nasko Apostolov, Nelson Evbarunegbe, & Raymond Li

Introduction

In the field of machine learning, **hyperparameter tuning** is the process of selecting an optimal set of **hyperparameters** for a learning algorithm.

Hyperparameters are user-selected parameters applied to learning algorithms that affect how they are implemented–for example, constraints, weights, learning rates, number of algorithm layers/complexity, etc.

Depending on which set of hyperparameters is selected, the results and performance of a particular machine learning algorithm can vary dramatically. The possible decision space, however, is enormous. In many cases, there are essentially an infinite set of possible combinations for all of the tuning parameters in a model. It is *impossible* to try every single one of them and find the absolute best option–instead, users must select a *good* option.

Typically, in hyperparameter tuning, users will pre-define a list of combinations to try. For example, if they must provide two parameters, β and λ, they might give the following potential values for these two parameters and ask the algorithm to try every possible combination within:

- $\beta = \{3, 5, 7, 9\}$
- $\lambda = \{0.001, 0.01, 0.1, 1.0\}$

This technique is known as a **grid search**.

It can sometimes produce decent results, but has several downsides.

1. The user must themselves define which lists of values to try. Coming up with this list introduces a high amount of subjectivity.
2. It is very computationally intensive (and slow) to try every single combination, one at a time. As the number of parameters increases, the number of combinations grows exponentially.
3. There is no easy way to compare results without a lot of additional coding on top of constructing the algorithm.

This is where hyperparameter tuning tools come into play. Several tools have been developed in recent years to help simplify this process and produce better results. Some use parallel computing to test a parameters simultaneously and speed up the training process, Many have UI implementations that can help make it easier for users to compare performance across models. Additionally, modern tuning software often provides advanced techniques to help the user select the best hyperparameters, such as intelligently moving the tuning in a direction that appears to be associated with increased performance scores scores.

Hyperopt (released in 2011) and **Optuna** (released in 2020) and are two industry standard parameter optimization tools designed to integrate with Python.

Hyperopt

Hyperopt is a popular Python library for hyperparameter optimization that has been around–and been considered industry standard–for more than a decade.

It is based on Bayesian optimization techniques, specifically Sequential Model-Based Optimization (SMBO). In SMBO, trials are run one after another, with each trial updating its hyperparameters by updating a probability model and applying Bayesian reasoning. Its goal is to maximize some "score" by modifying a configuration, and it does this by working to narrow the search space and use advanced algorithms for finding which combination of parameters can maximize the probability model.

Within SMBO, it implements several advanced hyperparameter tuning methods including:

- Tree-Structured Parzen Estimator (TPE)
- Adaptive Tree of Parzen Estimators (ATPE)
- Gaussian Processes (GP)

Hyperopt does not have a full UI implementation, but it does have several built in visualization functions for comparing models.

Hyperopt's code is publicly available on GitHub:

- https://github.com/hyperopt/hyperopt

It also has a basic documentation website:

- http://hyperopt.github.io/hyperopt/

Optuna

Optuna was released into the hyperparameter tuning market at a time when there were already quite a few established and trusted tools available (including the second tool we will discuss– Hyperopt).

Its creators chose to introduce Optuna after they discovered limitations in many other existing tuning algorithms, including that they:

1. Had instability in some environments
2. Were not utilizing the newest technology/advancements in hyperparameter optimization
3. Did not provide a way to specify which hyperparameters should be tuned directly within the Python code (instead, requiring the user to write special code just for the optimizer)

The creators of Optuna attempted to fill those gaps, advertising the following features:

1. **Define-By-Run style API** – An API style that is beginning to become industry standard in deep learning, but had not yet been applied to hyperparameter tuning prior to Optuna. Compared to an older system Define-And-Run. Using Define-By-Run allows users to write more modular code and access more complex hyperparameter spaces.
2. **Use of learning curves to prune trials** – Optuna uses deep learning and gradient boosting algorithms to predict the end result of a training trial before it is over. This can help it quit unpromising trials before they complete, improving efficiency.
3. **Parallel distributed optimization** – Optuna supports distributed optimization, simultaneously running multiple trials. This can dramatically speed up the tuning process and allow the user to increase the scale of how many parameters they can try.
4. **Visualization dashboard** - Optuna provides a UI dashboard where users can watch the optimization process and easily compare results from optimization experiments.

Like Hyperopt, the tool uses several advanced algorithms (including Bayesian methods) for tuning that go beyond standard grid search:

- Tree-Structured Parzen Estimator (TPE)
- Gaussian Processes (GP)
- Covariance Matrix Adaptation (CMA)
- Asynchronous Successive Halving Algorithm (ASHA)

These algorithms are similar to those used in Hyperopt, though Hyperopt is missing CMA and ASHA. The inclusion of ASHA in Optuna is particularly of note, as this is a key feature for allowing parallel optimization.

Optuna is framework agnostic, meaning it can easily be integrated with any of Python's machine learning/deep learning frameworks: Scikit-Learn, PyTorch, Tensorflow, etc.

The code for Optuna is publicly available on GitHub:

- https://github.com/optuna/optuna

There is a second GitHub page supporting their UI dashboard component, which was released separately:

- https://github.com/optuna/optuna-dashboard

Optuna also has a custom website with tutorials, examples, and links to documentation:

- https://optuna.org

In this project, we compare these two tools. We start with a code review of both tools based on the code available in their public repositories. After this, we then conduct an experimental comparison where we train and tune a machine learning model using Scikit-Learn in combination with both Hyperopt and Optuna and compare their performance.

Code Review: Hyperopt

Repository Link: GitHub

Repository Overview

In this section, we cover several basic characteristics of the GitHub repository that are not code specific. For each category, we rate the repository on a scale of 1-10.

Note: All of the information in this section is as of March 2023, and may have changed since then.

Popularity/Use

Hyperopt has over 6.6K stars, 1K forks, and 9.2K active GitHub users. 90% of repositories have $<$ 5K stars, putting Hyperopt in the top 10%. It clearly is having significant impact and influence in the field of hyperparameter tuning.

Score: 8/10

Community

Hyperopt has a reasonably active community with 90 separate contributors. There are 16 open pull requests from 13 separate users. This indicates that users *other* than those who originally created the project are helping maintain and improve the code, which can be helpful in keeping long-term projects updated.

Score: **7/10**

Maintenance

The last commit on the main branch is from Nov 29, 2021, nearly 1.5 years ago. The latest official release is from around the same time period (Nov 17, 2021).

Contrary to best practice, the `dev` branch is *not* updated compared to the `main` branch. It is 75 commits behind, when standard practice would be for the `dev` branch to be *ahead*(for testing out new features).

There are several pull requests that have been sitting in the repository for a long time, including one as old as August 2020.

Similarly, there are 375 open issues with a few dozen being **more than a decade old**!!

This indicates a somewhat poor level of maintenance.

Issues are getting buried. Pull requests are left opened instead of closed (if not going to be put into use/merged). Additionally, the longer that packages go without an update, the higher the likelihood for bugs to be introduced as dependencies and Python versions update and change. In the world of software, 1.5 years is a long time to not update.

Score: 4/10

Dependencies

Hyperopt has **8** direct dependencies, which is a relatively small number, especially for the scale of the project. The low number of dependencies helps keep the project stable by reducing the chance that a dependency could have vulnerabilities or that an update to a could introduce unexpected functionality-breaking bugs.

Score: 8/10

Security

A Snyk analysis found 0 direct vulnerabilities in the code and 3 indirect vulnerabilities, all associated with software dependencies.

These indirect vulnerabilities were classified as **Low** severity.

Overall, this indicates that Hyperopt is quite secure.

Score: 9/10

Code Review: Optuna

Experimental Comparison

Conclusion