**Methods in Hyperparameter Optimization**

**Daniel J. Donoghue Jr.**

MS Candidate at Johns Hopkins Whiting School of Engineering in Computational and Applied Mathematics. ddonogh1@jh.edu

*A computer program is said to learn from experience E with respect to some class of tasks T and performance measure*

*P if its performance at task in T, as measured by P, improves with experience E. – Tom M. Mitchell*

# Abstract

Machine learning models have an ever-increasing range of applications, ranging from reading tweets to gather the public perception of a politician, to creating statistical

models to predict stock market movements, to classifying

images in real-time to create a self-driving car experience. The models used in these tasks vary in size and complexity, but all have some key components, and run into some of the same issues.

In this paper, we will be exploring Hyperparameter Optimization for Machine Learning. The paper will begin

with a brief introduction to the topic and the motivations for doing this type of analysis. The next section will be a

discussion of some of the more prevalent methods of Hy-

perparameter Optimization today. The paper will conclude

by testing several of the most popular Hyperparameter Optimization python libraries on binary classification datasets.

# Introduction

In Machine Learning, hyperparameters are the inputs to the algorithm that are set before the learning process begins. This term is used in contrast to ‘parameters’, which will adjust while the learning is underway. As an example, in a Neural Network, things like the learning rate, the number of epochs, the number of Neurons per layer and the number of layers would be considered hyperparameters. They are set up in the system’s initial construction and do not change as the algorithm runs its course. Things like bias and weight values, which change as the neural network trains and updates them, are not considered hyperparameters. [3]

While many algorithms in machine learning packages come with default hyperparameters, it has been shown that better results often come if the hyperparameters are optimized for the specific problem/dataset. [2] There does not exist to date a ‘silver bullet’ hyperparameter set that works for all datasets, although rules-of-thumb are acceptable in some situations.

In addition to increasing accuracy, there are several other incentives to use hyperparameter optimization techniques to properly tune a model. First, less human effort and domain expertise is needed to design the machine learning algorithm. Also, using HPO tuning methods allows for scientific studies to compare methods more accurately. Properly tuning the hyperparameters of an algorithm is known to dramatically increase performance, and fair comparisons are more likely if the models are all properly tuned.[1]

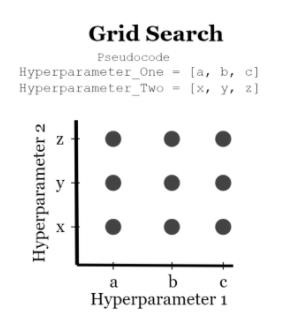
## Approaches for Hyperparameter Optimization

### Unguided Search Methods

The first and simplest approach to hyperparameter optimization is the exhaustive search. This process involves testing the entire set of all possible hyperparameter combinations.

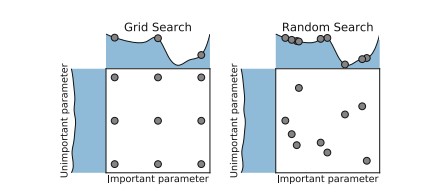
While this option, by its nature, is guaranteed to get you the most optimal set of the hyperparameters, it stands to reason that there may be significantly faster ways to achieve this result. This option may be acceptable if there are a small number of parameters with a very limited number of options. Since there are as many combinations as the product of the number of choices for each hyperparameter, checking every combination can get computationally inefficient very quickly.

Possibly the next most intuitive method for searching for optimal hyperparameters is the grid search. A grid search will begin with a maximum number of points to check, *p*. As visualized in Figure 1 below, a grid search will uniformly distribute it searches across the possible values for each hyperparameter. As such, the search points are set up to maximize the cartesian distance between each point from its closest neighbor. [4]



This method is intuitive and easy to implement, however it comes with some drawbacks. In an instance where changing one of the hyperparameters does not have much effect on the performance of the algorithm, valuable time can be wasted on iterations adjusting that parameter when that effort could instead be concentrated elsewhere. Additionally, if additional hyperparameter tuning is still determined to be needed after points p have all been checked, there is no easy way to continue the a grid search following the same methodology used initially.

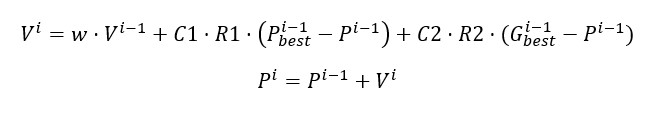
Another simple approach is the random search method. Much like the name implies, the algorithm begins by randomly selecting values for each hyperparameter and testing them while recording which combinations work best. Since grid search will only search p1/N (where N is the number of hyperparameters) values for each hyperparameter, while the random search will try up to p different values, it stands to reason that there may be performance benefits. This method has shown statistically significant improvements over a grid search whenever one hyperparameter has a disproportionate effect on the outcome of a model.[1] Figure 2 demonstrates a simple two hyperparameter problem where you are attempting to minimize and objective function. The x-axis parameter has significant effect on the objective function, while the parameter on the y-axis is relatively unimportant. Grid search only allows for a search of 3 points for the unimportant parameter, while random search allows for 9. As such, a much more optimal value is found.



### Guided Search Methods

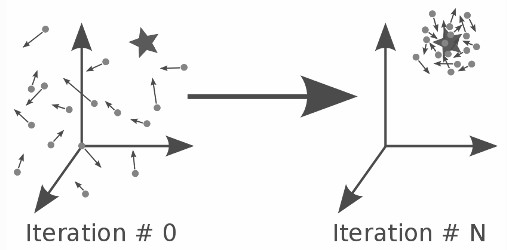
The methods described in the previous section were all ‘unguided’, meaning the results of the performance of one set of hyperparameters does not impact which points will next be chosen for testing. In contrast, guided search functions will start with an arbitrary configuration, test the results, adjust the hyperparameters, and repeat. If an improvement is made, the hyperparameters will continue to be adjusted in the same manner. If not, the changes will revert to the previous set and a different adjustment will be made. [1] Allowing the search to adapt will often increase the performance of the hyperparameter search method when compared to a grid search or a random search. [5] A popular class of guided methods are the populationbased methods. These include genetic algorithms, evolutionary algorithms, particle swarm optimization and more. Their behaviors are set to model that of biological communities by maintaining a population of configurations and perturbing it to obtain a newer, stronger population. [1] Particle Swarm Optimization (PSO) serves as a logical next step from a random search.

Since no prior knowledge of the function to be optimized exists, the particles are assigned a random position P (read as set of hyperparameters) and velocity V (to be interpreted as the rate in which the particles ‘flock’ towards the currently optimal set, PBest. Inertia weights w, acceleration coefficients C1 and C2, and randomly generated stochastic components R1 and R2 round out the equation below, which determines the rate of movement of particles across the spectrum of hyperparameters. [24]



As the PSO algorithm progresses, these weights and acceleration coefficients change. These adjustments are crucial to make sure that 1) the particles are moving fast enough to reach the optimal combination 2) the particles are not weighed down so much that they leave greener pastures ungrazed, 3) the particles have enough staying power to converge. As a result, with each passing generation of particles, they will move closer and closer across the spectrum of possible combinations towards a minimal value. From this description, it should be easy to draw the comparison between this optimization method and a flock of birds or swarm of bees closing in on food. [6]

PSO and other genetic algorithms repeat in a cycle much like that described above, until an optimal point is found, or the maximum number of iterations is reached. [6] Below is an intuitive visualization of Particle Swarm Optimization taken from the pagmo Github. [29]



Another budding approach is incorporating Bayesian Optimization to hyperparameter selection. Bayesian optimization has a ready found uses in A/B Testing, Recommendation Systems, Robotics and Reinforcement Learning, Environmental Monitoring and Sensor Networks, Preference Learning ad Interactive Interfaces, Automatic Machine Learning and Hyperparameter optimization, Combinatorial

Optimization and Natural Language Processing. [7] The concept of Bayesian Optimization centers around the of maximizing objective function *f* such that:



where x represents a set of hyperparameters in the domain space of interest, *X.* Objective function *f* represents the performance of the model at hyperparameter combination x and is not known. It can only be estimated using point-wise observations *y.* With each new instance of *y*, Bayesian Posterior updating is used to adjust the estimation of the objective function. With the updated version of *f*(x), the model will then a new xn+1 to test, in hopes of achieving the maximal value of x. [7]

When determining xn+1 the model is forced to trade off between exploration and exploitation. Exploration is defined as “attempting to discover new features about the world by selecting a suboptimal action” and exploitation is defined as “using what we already know about the world to get the best results we know of”. [13] In the context of this problem this can be thought of the decision to test hyperparameter combinations close to the best performing set already searched vs. exploring combinations that are very different from what has already been tested, in order to broaden the depth of the search.

This process is repeated until some criteria (usually an achieved level of accuracy or a maximum number of trials) is reached. [8] At this point, the model will return the best performing set of hyperparameters, x\*, given the model’s training.

## Python Packages for Hyperparameter Optimization

In this section we will review 3 different Hyperparameter Optimizing packages for Python. To test these packages, a variety of Binary Classification datasets were used. They include the ‘Advertising’ dataset from Kaggle [23], the ‘Adult’ dataset from the University of California [20], Irvine library, and the ‘Chess’ dataset, also from the UCI library. Data manipulation techniques like OneHotEncoding were used to get the data into a form that the algorithms could take. Each of the tuning methods were fed data in the same format.

|  |  |  |  |
| --- | --- | --- | --- |
| **Dataset**  **Name** | **Number of**  **Instances** | **Number of Variables** | **Positivity Rate** |
| Advertising | 1,000 | 242 | 50.0% |
| Adult | 32,561 | 108 | 24.1% |
| Chess | 3,196 | 73 | 52.2% |

These datasets were chosen for their simple formats and their varying degrees of size, allowing for a greater understanding of the ‘fixed costs’ of running an algorithm and the

‘variable costs’ of increases in the number of attributes and instances. Other binary classification datasets were tested, including Kaggle’s ‘Titanic’ and ‘Credit Card Fraud Detection’ datasets. [19,22]. ‘Titanic’ and similar datasets of that size were excluded from comparisons because they either train too quickly or do not have enough instances to create a noticeable variety in outcomes. ‘Credit Card Fraud Detection’ and similar datasets were excluded due to imbalances in their classifications. In this paper, the models were optimized for accuracy, whereas recall may have been more appropriate for these types of datasets. In this case, the models seemed to have determined that, seemingly regardless of the hyperparameters, predicting ‘not fraud’ for every instance leads to an accuracy of just over 99.8%. It appears the models deemed that good enough.

For a fair comparison, each of the hyperparameter training tools was trained on a Random Forest Classifier. The hyperparameters being tuned by the algorithm were the maximum depth of the tree, the maximum number of features on the tree, the criterion (a function to measure the quality of the split), and the number of trees in the forest. [21]

The datasets were split (with the same random state) at a rate of 70% used for training the model and 30% used for testing. The algorithms were scored on accuracy (the number of instances in the test sample predicted correctly/the total number of instances in the test sample). The amount of time it took to train and test the models was also recorded. For another unit of comparison, Sci-kit learn’s RandomizedSearchCV was also deployed to compare the results of the hyperparameter optimizing algorithm to a simple random search across the same domain of hyperparameters. Sklearn’s RandomForestClassifier was also tested with its default hyperparameter settings, to show if the hyperparameter tuning was having any effect on the outcome. Each algorithm, including the random search, was given 50 iterations to find a the best performing hyperparameter combination. No stopping functions were used, other than the limit on iterations. In the event of a rerun of this experiment, I would have included a function that force stopped the algorithm from continuing once either a certain accuracy was reached or a predetermined number of iterations occurred with no improvement in accuracy. This not only would have cut down on the computational time of the testing, but it would have given a more realistic estimate of how long it took the algorithms to get to the desired number, should they not have needed all 50 iterations.

### HyperOpt

Introduced in 2014, HyperOpt library offers optimization algorithms for search spaces that arise in algorithm configuration. It offers capabilities for not only hyperparameter selection in a given algorithm, but even algorithm selection tools as well. It supports a wide variety of variable types (categorical, ordinal, continuous, discrete), different sensitivity profiles (uniform vs logarithmic scaling) and conditional structures. [14]

In our use case of hyperparameter optimization, three inputs are required. These are the search domain (the possible values to be used at each hyperparameter), an objective function (a function f(x) such that the minimal value of f(x) represents the best performance of the algorithm), and an optimization algorithm. [14] At the time of this writing, the Hyperopt library supports three options for the optimization algorithm choice, Random Search, Tree of Parzen Estimators (TPE), and Adaptive TPE. [15] The Random Search follows the structure mentioned in the earlier section of this paper. In my testing, I used the default parameter for HyperOpt TPE. TPE’s are similar to Bayesian Optimization algorithms in that they rely on information from previous trials to propose better hyperparameters for the next evaluation. [26] Additional information about

TPE’s can be found in the sources below. [27]

### Optuna [25]

Optuna is the newest of the three libraries tested, which may explain its increased efficiency relative to its peers. Optuna is r simple to code in, taking only an objective function and the number of trials before performing optimization. It combines both relational sampling and independent sampling methods in hyperparameter selection. The relational sampling exploits the correlations among the parameters and the independent sampling allows for more independent parameter selection. The library will combine two algorithms like TPE (used in HyperOpt) and a Covariance Matrix Adaptation Evolution Strategy (CMA-ES). By incorporating more than one known tactic, Optuna scales relatively well compared to its peers. [28]

### Scikit Optimize BayesSearchCV [10]

BayesSearchCV is part of the popular scikit-optimize library. It is a relatively simple application of the Bayesian optimization of hyperparameters as described in section 1 of this paper.

### Testing and Conclusions

|  |  |  |
| --- | --- | --- |
| **Advertising Dataset** | |  |
| **Method** | **Accuracy** | **Time (in Seconds)** |
| Random Forest (No Hy-  perparameter Tuning) | 95.33% | 0.13 |
| Random Search | 96.00% | 29.12 |
| Optuna | 96.71% | 102.28 |
| HyperOpt | 96.00% | 147.48 |
| BayesSearchCV | 95.67% | 419.86 |

|  |  |  |
| --- | --- | --- |
| **Chess Dataset** | |  |
| **Method** | **Accuracy** | **Time (in Seconds)** |
| Random Forest (No Hy-  perparameter Tuning) | 98.96% | 0.15 |
| Random Search | 99.48% | 18.27 |
| Optuna | 99.28% | 111.97 |
| HyperOpt | 98.96% | 153.37 |
| BayesSearchCV | 99.58% | 670.54 |

|  |  |  |
| --- | --- | --- |
| **Adult Dataset** | |  |
| **Method** | **Accuracy** | **Time (in Seconds)** |
| Random Forest (No Hyperparameter Tuning) | 85.98% | 1.83 |
| Random Search | 86.48% | 528.16 |
| Optuna | 86.34% | 1153.11 |
| HyperOpt | 86.20% | 8120.31 |
| BayesSearchCV | 86.50% | 6870.85 |

I was surprised to see how minimal the improvement of the optimizers was, if there even was any. In the case of HyperOpt, it would also come at a dramatic time increase. It goes without saying that the problem was relatively constrained. Random Forests may simply not be as sensitive to tuning as other, more complicated algorithms. An untrained Random Forest also performed well on these datasets, which begs the question of if more ‘challenging’ datasets for the algorithms to accurately predict may have altered the results. Of the packages I tested, Optuna was the best performer, but still did not improve a random search in a statistically significant way, while taking substantially more computing time. Since a Random Search can happen to land on the optimal solution at any given point, it is not surprising that it could outperform the more coordinated models, but I was surprised to see it consistently matching HyperOpt, despite the dramatic increase in computing time. To follow up on this paper, more algorithms and datasets could be tested. Even Random Forest offer a few more hyperparameters to be configured, so there may even be improvement on this work. Expanding to Neural Networks, where the parameters may have a much larger effect on performance, also serves as a logical next testing point.

**References**

1. Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). Automated Machine Learning: Methods, Systems, Challenges. Springer.
2. Kohavi, R., & John, G. (2014, June 27). Automatic parameter selection by minimizing estimated error. Retrieved March 6, 2021, from https://www.sciencedirect.com/science/article/pii/B9781558603776500451 [3] DeepAI. (2019, May 17). Hyperparameter. Retrieved March 01, 2021, from https://deepai.org/machine-learningglossary-and-terms/hyperparameter
3. Stalfort, J. (2019, June 06). Hyperparameter tuning using grid search and random search. Retrieved March 4, 2021, from [https://medium.com/@jackstalfort/hyperparametertuning-using-grid-search-and-random-searchf8750a464b35](https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35)
4. Li, K., Malik, J. (2017, November 30). Learning to optimize neural nets. Retrieved March 4, 2021, from https://arxiv.org/pdf/1703.00441
5. Escalante, H. J., Montes, M.,; Sucar, L. E. (2009, February). Particle Swarm Model Selection. Retrieved March 01, 2021, from <https://jmlr.csail.mit.edu/papers/volume10/escalante09a/escalante09a.pdf>
6. Shahriari, B., Swersky, K., Wang, Z., Adams, R., De Freitas, N. (1970, January 01). Taking the human out of the loop: A review of bayesian optimization. Retrieved March 06, 2021, from <https://dash.harvard.edu/handle/1/27769882>
7. Gangopadhyay, B. (2018, October 01). Bayesian optimization for dummies. Retrieved March 7, 2021, from [https://www.linkedin.com/pulse/bayesian-optimizationdummies-briti-gangopadhyay/](https://www.linkedin.com/pulse/bayesian-optimization-dummies-briti-gangopadhyay/)
8. Bergsta, J., Bengio, Y., Bardenet, R., Kegl, B. (2011).

Algorithms for HYPER-PARAMETER OPTIMIZATION.

Retrieved March 9, 2021, from <https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf>[10] Scikit Optimize BayesSearchCV. (2020). Retrieved March 13, 2021, from <https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html#skopt.BayesSearchCV>

1. Kohavi, R., John, G. (2014, June 27). Automatic parameter selection by minimizing estimated error. Retrieved March 6, 2021, from <https://www.sciencedirect.com/science/article/pii/B9781558603776500451>
2. 2021. (n.d.). Ray v1.2.0. Retrieved March 11, 2021, from <https://docs.ray.io/en/latest/tune/index.html>[13] Coggan, M. (2004). Exploration and exploitation in reinforcement learning. Retrieved March 19, 2021, from <http://www.archive2.cra.org/Activities/craw_archive/dmp/awards/2004/Coggan/FinalReport.pdf>[14] Komer, B., Bergsta, J., Eliasmith, C. (2014). HyperoptSklearn: Automatic Hyperparameter Configuration for Scikit-Learn. Retrieved March 2, 2021, from <http://compneuro.uwaterloo.ca/files/publications/komer.2014b.pdf>[15] Hyperopt: Distributed Asynchronous Hyper-parameter Optimization. (n.d.). Retrieved April 01, 2021, from <http://hyperopt.github.io/hyperopt/>
3. Bergstra, J., Yamins, D., Cox, D. D. (2013) Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. To appear in Proc. of the 30th International Conference on Machine Learning (ICML 2013).
4. Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J., Stoica, I. (2018, July 13). Tune: A research platform for distributed model selection and training. Retrieved March 21, 2021, from <https://arxiv.org/abs/1807.05118>[18] Ray v1.2.0. (2021). Retrieved April 1, 2021, from <https://docs.ray.io/en/latest/tune/api_docs/suggestion.html#tune-search-alg>
5. Titanic - machine learning from disaster. (n.d.). Retrieved March 04, 2021, from <https://www.kaggle.com/c/titanic>
6. Kohavi, R., Becker, B. (1996). UCI machine Learning Repository: Adult data set. Retrieved March 21, 2021, from <https://archive.ics.uci.edu/ml/datasets/adult>[21] Sklearn.ensemble.RandomForestClassifier. (n.d.). Retrieved March 28, 2021, from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
7. ULB, M. (2018, March 23). Credit card fraud detection.

Retrieved March 19, 2021, from

<https://www.kaggle.com/mlg-ulb/creditcardfraud>

1. Fayomi. (2017, December 14). Advertising. Retrieved April 01, 2021, from <https://www.kaggle.com/fayomi/advertising>
2. Buchacher, T. (2020, April 30). Hyperparameter Optimization Utilizing a Particle Swarm Approach. Retrieved April 01, 2021, from [https://medium.com/next-level-german-engineering/hyperparameter-optimisation-utilising-aparticle-swarm-approach-5711957b3f3f](https://medium.com/next-level-german-engineering/hyperparameter-optimisation-utilising-a-particle-swarm-approach-5711957b3f3f)
3. David, D. (2020, October 12). Hyperparameter optimization techniques to improve your machine LEARNING MODEL'S PERFORMANCE. Retrieved April 01, 2021, from [https://www.freecodecamp.org/news/hyperparameteroptimization-techniques-machine-learning/](https://www.freecodecamp.org/news/hyperparameter-optimization-techniques-machine-learning/) [26] Koehrsen, W. (2018, July 02). A conceptual explanation of bayesian hyperparameter optimization for machine learning. Retrieved April 01, 2021, from <https://towardsdatascience.com/a-conceptual-explanation-of-bayesian-model-based-hyperparameter-optimization-for-machine-learning-b8172278050f>
4. Hoang-Phuong Nguyen, Jie Liu, Enrico Zio. A longterm prediction approach based on long short-term memory neural networks with automatic parameter optimization by Tree-structured Parzen Estimator and applied to time-series data of NPP steam generators. Applied Soft Computing, Elsevier, 2020, 89, pp.106116. ff10.1016/j.asoc.2020.106116ff. ffhal02470820
5. Akiba, T., Sano, S., Yanase, T., Ohta, T.,; Koyama, M. (2019, July 25). Optuna: A Next-generation hyperparameter optimization framework. Retrieved April 03, 2021, from <https://arxiv.org/abs/1907.10902>
6. Particle swarm optimization (pso). (n.d.). Retrieved

April 01, 2021, from

https://esa.github.io/pagmo2/docs/cpp/algorithms/pso.html