WIKIPEDIA
The Free Encyclopedia

Hyperparameter optimization

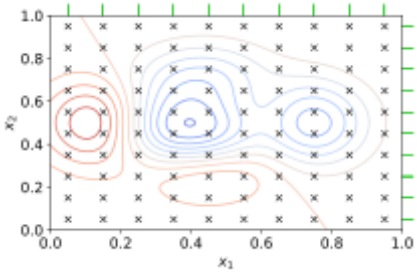
In machine learning, **hyperparameter optimization**[1] or tuning is the problem of choosing a set of optimal hyperparameters for a learning algorithm. A hyperparameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are learned.

The same kind of machine learning model can require different constraints, weights or learning rates to generalize different data patterns. These measures are called hyperparameters, and have to be tuned so that the model can optimally solve the machine learning problem. Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given independent data.[2] The objective function takes a tuple of hyperparameters and returns the associated loss.[2] Cross-validation is often used to estimate this generalization performance, and therefore choose the set of values for hyperparameters that maximize it.[3]

Approaches

Grid search

The traditional way of performing hyperparameter optimization has been *grid search*, or a *parameter sweep*, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set[4] or evaluation on a hold-out validation set.[5]



Grid search across different values of two hyperparameters. For each hyperparameter, 10 different values are considered, so a total of 100 different combinations are evaluated and compared. Blue contours indicate regions with strong results, whereas red ones show regions with poor results.

Since the parameter space of a machine learner may include real-valued or unbounded value spaces for certain parameters, manually set bounds and discretization may be necessary before applying grid search.

For example, a typical soft-margin SVM classifier equipped with an RBF kernel has at least two hyperparameters that need to be tuned for good performance on unseen data: a regularization constant C and a kernel hyperparameter γ. Both parameters are continuous, so to perform grid search, one selects a finite set of "reasonable" values for each, say

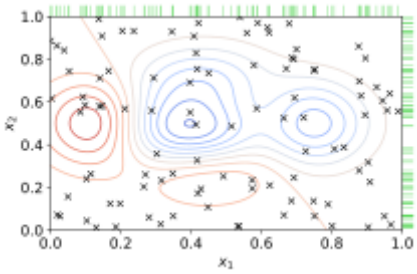
$$C \in \{10, 100, 1000\}$$
$$\gamma \in \{0.1, 0.2, 0.5, 1.0\}$$

Grid search then trains an SVM with each pair (C, γ) in the Cartesian product of these two sets and evaluates their performance on a held-out validation set (or by internal cross-validation on the training set, in which case multiple SVMs are trained per pair). Finally, the grid search algorithm outputs the settings that achieved the highest score in the validation procedure.

Grid search suffers from the curse of dimensionality, but is often embarrassingly parallel because the hyperparameter settings it evaluates are typically independent of each other.[3]

Random search

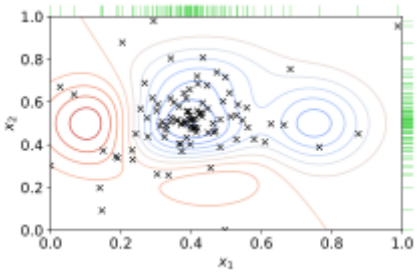
Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm.[3] In this case, the optimization problem is said to have a low intrinsic dimensionality.[6] Random Search is also embarrassingly parallel, and additionally allows the inclusion of prior knowledge by specifying the distribution from which to sample. Despite its simplicity, random search remains one of the important base-lines against which to compare the performance of new hyperparameter optimization methods.



Random search across different combinations of values for two hyperparameters. In this example, 100 different random choices are evaluated. The green bars show that more individual values for each hyperparameter are considered compared to a grid search.

Bayesian optimization

Bayesian optimization is a global optimization method for noisy black-box functions. Applied to hyperparameter optimization, Bayesian optimization builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a validation set. By iteratively evaluating a promising hyperparameter configuration based on the current model, and then updating it, Bayesian optimization aims to gather observations revealing as much information as possible about this function and, in particular, the location of the optimum. It tries to balance exploration (hyperparameters for which the outcome is most uncertain) and exploitation (hyperparameters expected close to the optimum). In practice, Bayesian optimization has been shown[7][8][9][10] to obtain better results in fewer evaluations compared to grid search and random search, due to the ability to reason about the quality of experiments before they are run.



Methods such as Bayesian optimization smartly explore the space of potential choices of hyperparameters by deciding which combination to explore next based on previous observations.

Gradient-based optimization

For specific learning algorithms, it is possible to compute the gradient with respect to hyperparameters and then optimize the hyperparameters using gradient descent. The first usage of these techniques was focused on neural networks.[11] Since then, these methods have been extended to other models such as support vector machines[12] or logistic regression.[13]

A different approach in order to obtain a gradient with respect to hyperparameters consists in differentiating the steps of an iterative optimization algorithm using automatic differentiation.[14][15][16][17] A more recent work along this direction uses the implicit function theorem to calculate hypergradients and proposes a stable approximation of the inverse Hessian. The method scales to millions of hyperparameters and requires constant memory.

In a different approach,[18] a hypernetwork is trained to approximate the best response function. One of the advantages of this method is that it can handle discrete hyperparameters as well. Self-tuning networks[19] offer a memory efficient version of this approach by choosing a compact representation for the hypernetwork. More recently, Δ-STN[20] has improved this method further by a slight reparameterization of the hypernetwork which speeds up training. Δ-STN also yields a better approximation of the best-response Jacobian by linearizing the network in the weights, hence removing unnecessary nonlinear effects of large changes in the weights.

Apart from hypernetwork approaches, gradient-based methods can be used to optimize discrete hyperparameters also by adopting a continuous relaxation of the parameters.[21] Such methods have been extensively used for the optimization of architecture hyperparameters in neural architecture search.

Evolutionary optimization

Evolutionary optimization is a methodology for the global optimization of noisy black-box functions. In hyperparameter optimization, evolutionary optimization uses evolutionary algorithms to search the space of hyperparameters for a given algorithm.[8] Evolutionary hyperparameter optimization follows a process inspired by the biological concept of evolution:

1. Create an initial population of random solutions (i.e., randomly generate tuples of hyperparameters, typically 100+)
2. Evaluate the hyperparameters tuples and acquire their fitness function (e.g., 10-fold cross-validation accuracy of the machine learning algorithm with those hyperparameters)
3. Rank the hyperparameter tuples by their relative fitness
4. Replace the worst-performing hyperparameter tuples with new hyperparameter tuples generated through crossover and mutation
5. Repeat steps 2-4 until satisfactory algorithm performance is reached or algorithm performance is no longer improving

Evolutionary optimization has been used in hyperparameter optimization for statistical machine learning algorithms,[8] automated machine learning, typical neural network [22] and deep neural network architecture search,[23][24] as well as training of the weights in deep neural networks.[25]

Population-based

Population Based Training (PBT) learns both hyperparameter values and network weights. Multiple learning processes operate independently, using different hyperparameters. As with evolutionary methods, poorly performing models are iteratively replaced with models that adopt modified hyperparameter values and weights based on the better performers. This replacement model warm starting is the primary differentiator between PBT and other evolutionary methods. PBT thus allows the hyperparameters to evolve and eliminates the need for manual hypertuning. The process makes no assumptions regarding model architecture, loss functions or training procedures.

PBT and its variants are adaptive methods: they update hyperparameters during the training of the models. On the contrary, non-adaptive methods have the sub-optimal strategy to assign a constant set of hyperparameters for the whole training.[26]

Early stopping-based

A class of early stopping-based hyperparameter optimization algorithms is purpose built for large search spaces of continuous and discrete hyperparameters, particularly when the computational cost to evaluate the performance of a set of hyperparameters is high. Irace implements the iterated racing algorithm, that focuses the search around the most promising configurations, using statistical tests to discard the ones that perform poorly.[27][28] Another early stopping hyperparameter optimization algorithm is successive halving (SHA),[29] which begins as a random search but periodically prunes low-performing models, thereby focusing computational resources on more promising models. Asynchronous successive halving (ASHA)[30] further improves upon SHA's resource utilization profile by removing the need to synchronously evaluate and prune low-performing models. Hyperband[31] is a higher level early stopping-based algorithm that invokes SHA or ASHA multiple times with varying levels of pruning aggressiveness, in order to be more widely applicable and with fewer required inputs.

Others

RBF[32] and spectral[33] approaches have also been developed.

Issues with hyperparameter optimization

When hyperparameter optimization is done, the set of hyperparameters are often fitted on a training set and selected based on the generalization performance, or score, of a validation set. However, this procedure is at risk of overfitting the hyperparameters to the validation set. Therefore, the generalization performance score of the validation set (which can be several sets in the case of a cross-validation procedure) cannot be used to simultaneously estimate the generalization performance of the final model. In order to do so, the generalization performance has to be evaluated on a set independent (which has no intersection) of the set (or sets) used for the optimization of the hyperparameters, otherwise the performance might give a value which is too optimistic (too large). This can be done on a second test set, or through an outer cross-validation procedure called nested cross-validation, which allows an unbiased estimation of the generalization performance of the model, taking into account the bias due to the hyperparameter optimization.

See also

- Automated machine learning
- Neural architecture search
- Meta-optimization
- Model selection
- Self-tuning
- XGBoost

References

1. Matthias Feurer and Frank Hutter. Hyperparameter optimization (https://link.springer.com/content/pdf/10.1007%2F978-3-030-05318-5_1.pdf). In: *AutoML: Methods, Systems, Challenges*, pages 3–38.
2. Claesen, Marc; Bart De Moor (2015). "Hyperparameter Search in Machine Learning". arXiv:1502.02127 (https://arxiv.org/abs/1502.02127) [cs.LG (https://arxiv.org/archive/cs.LG)].

3. Bergstra, James; Bengio, Yoshua (2012). "Random Search for Hyper-Parameter Optimization" (http://jmlr.csail.mit.edu/papers/volume13/bergstra12a/bergstra12a.pdf) (PDF). *Journal of Machine Learning Research*. **13**: 281–305.
4. Chin-Wei Hsu, Chih-Chung Chang and Chih-Jen Lin (2010). A practical guide to support vector classification (http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf). Technical Report, National Taiwan University.
5. Chicco D (December 2017). "Ten quick tips for machine learning in computational biology" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5721660). *BioData Mining*. **10** (35): 35. doi:10.1186/s13040-017-0155-3 (https://doi.org/10.1186%2Fs13040-017-0155-3). PMC 5721660 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5721660). PMID 29234465 (https://pubmed.ncbi.nlm.nih.gov/29234465).
6. Ziyu, Wang; Frank, Hutter; Masrour, Zoghi; David, Matheson; Nando, de Feitas (2016). "Bayesian Optimization in a Billion Dimensions via Random Embeddings". *Journal of Artificial Intelligence Research*. **55**: 361–387. arXiv:1301.1942 (https://arxiv.org/abs/1301.1942). doi:10.1613/jair.4806 (https://doi.org/10.1613%2Fjair.4806). S2CID 279236 (https://api.semanticscholar.org/CorpusID:279236).
7. Hutter, Frank; Hoos, Holger; Leyton-Brown, Kevin (2011), "Sequential model-based optimization for general algorithm configuration" (http://www.cs.ubc.ca/labs/beta/Projects/SMAC/papers/11-LION5-SMAC.pdf) (PDF), *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, **6683**: 507–523, CiteSeerX 10.1.1.307.8813 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.307.8813), doi:10.1007/978-3-642-25566-3_40 (https://doi.org/10.1007%2F978-3-642-25566-3_40), ISBN 978-3-642-25565-6, S2CID 6944647 (https://api.semanticscholar.org/CorpusID:6944647)
8. Bergstra, James; Bardenet, Remi; Bengio, Yoshua; Kegl, Balazs (2011), "Algorithms for hyper-parameter optimization" (http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf) (PDF), *Advances in Neural Information Processing Systems*
9. Snoek, Jasper; Larochelle, Hugo; Adams, Ryan (2012). "Practical Bayesian Optimization of Machine Learning Algorithms" (http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf) (PDF). *Advances in Neural Information Processing Systems*. arXiv:1206.2944 (https://arxiv.org/abs/1206.2944). Bibcode:2012arXiv1206.2944S (https://ui.adsabs.harvard.edu/abs/2012arXiv1206.2944S).
10. Thornton, Chris; Hutter, Frank; Hoos, Holger; Leyton-Brown, Kevin (2013). "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms" (http://www.cs.ubc.ca/labs/beta/Projects/autoweka/papers/autoweka.pdf) (PDF). *Knowledge Discovery and Data Mining*. arXiv:1208.3719 (https://arxiv.org/abs/1208.3719). Bibcode:2012arXiv1208.3719T (https://ui.adsabs.harvard.edu/abs/2012arXiv1208.3719T).
11. Larsen, Jan; Hansen, Lars Kai; Svarer, Claus; Ohlsson, M (1996). "Design and regularization of neural networks: the optimal use of a validation set" (http://orbit.dtu.dk/files/4545571/Svarer.pdf) (PDF). *Proceedings of the 1996 IEEE Signal Processing Society Workshop*: 62–71. CiteSeerX 10.1.1.415.3266 (https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.415.3266). doi:10.1109/NNSP.1996.548336 (https://doi.org/10.1109%2FNNSP.1996.548336). ISBN 0-7803-3550-3. S2CID 238874 (https://api.semanticscholar.org/CorpusID:238874).
12. Olivier Chapelle; Vladimir Vapnik; Olivier Bousquet; Sayan Mukherjee (2002). "Choosing multiple parameters for support vector machines" (http://www.chapelle.cc/olivier/pub/mlj02.pdf) (PDF). *Machine Learning*. **46**: 131–159. doi:10.1023/a:1012450327387 (https://doi.org/10.1023%2Fa%3A1012450327387).
13. Chuong B; Chuan-Sheng Foo; Andrew Y Ng (2008). "Efficient multiple hyperparameter learning for log-linear models" (http://papers.nips.cc/paper/3286-efficient-multiple-hyperparameter-learning-for-log-linear-models.pdf) (PDF). *Advances in Neural Information Processing Systems*. **20**.

14. Domke, Justin (2012). "Generic Methods for Optimization-Based Modeling" (https://web.archive.org/web/20140124182520/http://jmlr.org/proceedings/papers/v22/domke12/domke12.pdf) (PDF). *Aistats*. **22**. Archived from the original (http://www.jmlr.org/proceedings/papers/v22/domke12/domke12.pdf) (PDF) on 2014-01-24. Retrieved 2017-12-09.
15. Maclaurin, Douglas; Duvenaud, David; Adams, Ryan P. (2015). "Gradient-based Hyperparameter Optimization through Reversible Learning". arXiv:1502.03492 (https://arxiv.org/abs/1502.03492) [stat.ML (https://arxiv.org/archive/stat.ML)].
16. Franceschi, Luca; Donini, Michele; Frasconi, Paolo; Pontil, Massimiliano (2017). "Forward and Reverse Gradient-Based Hyperparameter Optimization" (http://proceedings.mlr.press/v70/franceschi17a/franceschi17a-supp.pdf) (PDF). *Proceedings of the 34th International Conference on Machine Learning*. arXiv:1703.01785 (https://arxiv.org/abs/1703.01785). Bibcode:2017arXiv170301785F (https://ui.adsabs.harvard.edu/abs/2017arXiv170301785F).
17. Shaban, A., Cheng, C. A., Hatch, N., & Boots, B. (2019, April). Truncated back-propagation for bilevel optimization (https://arxiv.org/pdf/1810.10667.pdf). In *The 22nd International Conference on Artificial Intelligence and Statistics* (pp. 1723-1732). PMLR.
18. Lorraine, J., & Duvenaud, D. (2018). Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*.
19. MacKay, M., Vicol, P., Lorraine, J., Duvenaud, D., & Grosse, R. (2019). Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *arXiv preprint arXiv:1903.03088*.
20. Bae, J., & Grosse, R. B. (2020). Delta-stn: Efficient bilevel optimization for neural networks using structured response jacobians. *Advances in Neural Information Processing Systems*, *33*, 21725-21737.
21. Liu, H., Simonyan, K., & Yang, Y. (2018). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.
22. Kousiouris G, Cuccinotta T, Varvarigou T (2011). "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks" (https://www.sciencedirect.com/science/article/abs/pii/S0164121211000951). *Journal of Systems and Software*. **84** (8): 1270–1291. doi:10.1016/j.jss.2011.04.013 (https://doi.org/10.1016%2Fj.jss.2011.04.013). hdl:11382/361472 (https://hdl.handle.net/11382%2F361472).
23. Miikkulainen R, Liang J, Meyerson E, Rawal A, Fink D, Francon O, Raju B, Shahrzad H, Navruzyan A, Duffy N, Hodjat B (2017). "Evolving Deep Neural Networks". arXiv:1703.00548 (https://arxiv.org/abs/1703.00548) [cs.NE (https://arxiv.org/archive/cs.NE)].
24. Jaderberg M, Dalibard V, Osindero S, Czarnecki WM, Donahue J, Razavi A, Vinyals O, Green T, Dunning I, Simonyan K, Fernando C, Kavukcuoglu K (2017). "Population Based Training of Neural Networks". arXiv:1711.09846 (https://arxiv.org/abs/1711.09846) [cs.LG (https://arxiv.org/archive/cs.LG)].
25. Such FP, Madhavan V, Conti E, Lehman J, Stanley KO, Clune J (2017). "Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning". arXiv:1712.06567 (https://arxiv.org/abs/1712.06567) [cs.NE (https://arxiv.org/archive/cs.NE)].
26. Li, Ang; Spyra, Ola; Perel, Sagi; Dalibard, Valentin; Jaderberg, Max; Gu, Chenjie; Budden, David; Harley, Tim; Gupta, Pramod (2019-02-05). "A Generalized Framework for Population Based Training". arXiv:1902.01894 (https://arxiv.org/abs/1902.01894) [cs.AI (https://arxiv.org/archive/cs.AI)].

27. López-Ibáñez, Manuel; Dubois-Lacoste, Jérémie; Pérez Cáceres, Leslie; Stützle, Thomas; Birattari, Mauro (2016). "The irace package: Iterated Racing for Automatic Algorithm Configuration" (https://doi.org/10.1016%2Fj.orp.2016.09.002). *Operations Research Perspective*. **3** (3): 43–58. doi:10.1016/j.orp.2016.09.002 (https://doi.org/10.1016%2Fj.orp.2016.09.002).
28. Birattari, Mauro; Stützle, Thomas; Paquete, Luis; Varrentrapp, Klaus (2002). "A Racing Algorithm for Configuring Metaheuristics". *Gecco 2002*: 11–18.
29. Jamieson, Kevin; Talwalkar, Ameet (2015-02-27). "Non-stochastic Best Arm Identification and Hyperparameter Optimization". arXiv:1502.07943 (https://arxiv.org/abs/1502.07943) [cs.LG (https://arxiv.org/archive/cs.LG)].
30. Li, Liam; Jamieson, Kevin; Rostamizadeh, Afshin; Gonina, Ekaterina; Hardt, Moritz; Recht, Benjamin; Talwalkar, Ameet (2020-03-16). "A System for Massively Parallel Hyperparameter Tuning". arXiv:1810.05934v5 (https://arxiv.org/abs/1810.05934v5) [cs.LG (https://arxiv.org/archive/cs.LG)].
31. Li, Lisha; Jamieson, Kevin; DeSalvo, Giulia; Rostamizadeh, Afshin; Talwalkar, Ameet (2020-03-16). "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". *Journal of Machine Learning Research*. **18**: 1–52. arXiv:1603.06560 (https://arxiv.org/abs/1603.06560).
32. Diaz, Gonzalo; Fokoue, Achille; Nannicini, Giacomo; Samulowitz, Horst (2017). "An effective algorithm for hyperparameter optimization of neural networks". arXiv:1705.08520 (https://arxiv.org/abs/1705.08520) [cs.AI (https://arxiv.org/archive/cs.AI)].
33. Hazan, Elad; Klivans, Adam; Yuan, Yang (2017). "Hyperparameter Optimization: A Spectral Approach". arXiv:1706.00764 (https://arxiv.org/abs/1706.00764) [cs.LG (https://arxiv.org/archive/cs.LG)].
