Chapter 6
Hyperparameter Tuning and Optimization Applications

Thomas Bartz-Beielstein

Abstract This chapter reflects on advantages and sense of use of Hyperparameter Tuning (HPT) and its disadvantages. In particular it shows how important it is, to keep the human in the loop, even if HPT works perfectly. The chapter presents a collection of HPT studies. First, HPT applications in Machine Learning (ML) and Deep Learning (DL) are described. A special focus lies on automated ML, neural architecture search, and combined approaches. HPT software is presented. Finally, model based approaches, especially applications with Sequential Parameter Optimization Toolbox (SPOT) are discussed.

6.1 Surrogate Optimization

Starting in the 1960s, Response Surface Methodology (RSM) and related Design of Experiments (DOE) methods were transferred from the engineering domain (e.g., from physics, agriculture, chemistry, and aerospace) to computer science (Montgomery 2017). With the increasing computational power, computer simulations, scientific computing and computational statistics gained importance (Gentle et al. 2004; Strang 2007). Kleijnen (1987) summarizes these ideas and methods in a very comprehensible manner for simulation practitioners. After computer simulations replaced expensive lab experiments, these computer simulations themselves were substituted by even cheaper computer models: surrogate models or in short, *surrogates*, that imitate complex numerical calculations, were developed. Kriging surrogates (or Gaussian Processs (GPs) aka Bayesian Optimization (BO)), that gleaned ideas from computer experiments in geostatistics, enjoy wide applicability, especially in domains where predictions are required. Today, GP models are used as powerful predictors for all sorts of applications in engineering and ML. GP methods replaced classical regression methods in many domains. Santner et al. (2003), Forrester et al. (2008b), and Gramacy (2020) wrote groundbreaking works in this field. In global optimization, Efficient Global Optimization (EGO) became a very popular approach

T. Bartz-Beielstein (✉)
Institute for Data Science, Engineering and Analytics, TH Köln, Cologne, Germany
e-mail: thomas.bartz-beielstein@th-koeln.de

E. Bartz et al. (eds.), *Hyperparameter Tuning for Machine and Deep Learning with R*,
https://doi.org/10.1007/978-981-19-5170-1_6

(Schonlau 1997; Jones et al. 1998). Emmerich (2005) showed that surrogates can significantly accelerate evolutionary multi-objective optimization algorithms in the presence of time consuming evaluations.

SPOT was one of the first approaches that combined classical regression, surrogate optimization (Kriging), and, especially for non-continuous variables, decision trees, for the optimization of algorithm (hyper-)parameters (Bartz-Beielstein et al. 2004, 2005). Applied in hyperparameter optimization in Evolutionary Computation (EC), see, e.g., Lobo et al. (2007), the collection *Experimental Methods for the Analysis of Optimization Algorithms* includes important publications in this field (Bartz-Beielstein et al., 2010). For example, the following contributions paved the way for important developments in HPT:

- Ridge and Kudenko (2010) describe the classical DOE approach, e.g., how to set up experimental designs, for algorithm benchmarking.
- The contribution *Sequential Model-Based Optimization* by Hutter et al. (2010b) laid the foundation for surrogate optimization in ML and resulted in software tools such as Sequential Model-Based Optimization for General Algorithm Configuration (SMAC) (Hutter et al. 2010a).
- Iterative Racing (IRACE), which is a generalization of the Iterated F-race method, is another popular tool for the automatic configuration of optimization algorithms (Birattari et al. 2009).

Surrogate optimization is the *de facto* standard for complex optimization problems, especially for continuous variables. Bartz-Beielstein and Zaefferer (2017) presented methods for continuous and discrete optimization based on Kriging. While surrogate models are well-established in the continuous optimization domain, they are less frequently applied to more complex search spaces with discrete or combinatorial solution representations (Zaefferer et al. 2014). Zaefferer (2018) Ph.D. thesis fills this gap, showing how surrogate models like Kriging can be extended to arbitrary problem types, if measures of similarity between candidate solutions are available. Today, surrogate optimization is applied in the engineering domain as well as in computer science, e.g., for HPT.

Example: Mixed-Discrete Problems

Many real-world optimization problems consider the optimization of ordinal integers, categorical integers, binary variables, permutations, strings, trees, or graphs structures in general. These real-world problems pose complex search spaces which require a deep understanding of the underlying solution representations.

Some of them, for example integers, are more suitable to be treated by classic optimization algorithms. Others, such as trees, have to be handled by specifically developed optimization algorithms. In general, solving these kinds of problems usually necessitates a significant number of objective function evaluations. However, in many engineering problems, a single evaluation is based on either on experimental or numerical analysis. This causes significant costs with respect to time or resources.

Surrogate Model Based Optimization (SMBO) aims to handle the complex variable structures and the limited budget simultaneously. Sequential Parameter Optimization (SPO) pursues the identification of global optima taking advantage of a budget allocation process that maximizes the information gain in promising regions. Gentile et al. (2021) presented an efficient method to face mixed-discrete optimization problems using surrogates.

Example: Alzheimer's Disease

Bloch and Friedrich (2021) used ML for early detection of Alzheimer's disease especially based on magnetic resonance imaging. The authors use BO to time-efficiently find good hyperparameters for Random Forest (RF) and Extreme Gradient Boosting (XGBoost) models, which are based on four and seven hyperparameters and promise good classification results. Those models are applied to distinguish if mild cognitive impaired subjects from the Alzheimer's disease neuroimaging initiative data set will prospectively convert to Alzheimer's disease.

The results showed comparable Cross Validation (CV) classification accuracies for models trained using BO and grid-search, whereas BO has been less time-consuming. Similar to the approaches presented in this book (and in many other BO studies), the initial combinations for BO were set using Latin Hypercube Design (LHD) and via random initialization. Furthermore, many models trained using BO achieved better classification results for the independent test data set than the model based on the grid-search. The best model was an XGBoost model trained with BO.

Example: Elevator Simulation and Optimization

Modern elevator systems are controlled by the elevator group controllers that assign moving and stopping policies to the elevator cars. Designing an adequate Elevator Group Control (EGC) policy is challenging for a number of reasons, one of them being conflicting optimization objectives. Vodopija et al. (2022) address this task by formulating a corresponding constrained multiobjective optimization problem, and, in contrast to most studies in this domain, approach it using true multiobjective optimization methods capable of finding approximations for Pareto-optimal solutions.

Specifically, they apply five multiobjective optimization algorithms with default constraint handling techniques and demonstrate their performance in optimizing EGC for nine elevator systems of various complexity. SPO was used to tune the algorithm parameters. The experimental results confirm the scalability of the proposed methodology and suggest that NSGA-II equipped with the constrained-domination principle is the best performing algorithm on the test EGC systems. The proposed problem formulation and methodology allow for better understanding of the EGC design problem and provide insightful information to the stakeholders involved in deciding on elevator system configurations and control policies.

Example: Cyber-physical Production Systems

Bunte et al. (2019) developed a cognitive architecture for Artificial Intelligence (AI) in Cyber-physical Production Systemss (CPPSs). The goal of this architecture is to reduce the implementation effort of AI algorithms in CPPSs. Declarative user goals and the provided algorithm-knowledge base allow the dynamic pipeline orchestration and configuration. A big data platform instantiates the pipelines and monitors the CPPSs performance for further evaluation through the cognitive module. Thus, the cognitive module is able to select feasible and robust configurations for process pipelines in varying use cases. Furthermore, it automatically adapts the models and algorithms based on model quality and resource consumption. The cognitive module also instantiates additional pipelines to evaluate algorithms from different classes on test functions.

Example: Resource Planning in Hospitals

Pandemics pose a serious challenge to health-care institutions. To support the resource planning of health authorities from the Cologne region, BaBSim.Hospital, a tool for capacity planning based on discrete event simulation, was created Bartz-Beielstein et al. (2021a). The predictive quality of the simulation is determined by 29 parameters with reasonable default values obtained in discussions with medical professionals. Bartz-Beielstein et al. (2021b) aimed to investigate and optimize these parameters to improve BaBSim.Hospital using a surrogate optimization approach and an in-depth sensitivity analysis.

Because SMBO is the default method in optimization via simulation, there are many more examples from the application domain, e.g., Waibel et al. (2019) present methods for selecting tuned hyper-parameters of search heuristics for computationally expensive simulation-based optimization problems.

6.2 Hyperparameter Tuning in Machine and Deep Learning

In contrast to HPT in optimization, where the objective function with related input parameters is clearly specified for the tuner, the situation in ML is more complex. As illustrated in Fig. 2.2, the tuner is confronted with several loss functions, metrics, and data sets. As discussed in Sect. 2.3, there is no clear answer to this problem.

Furthermore, the situation in ML is more challenging than in optimization, because ML methods develop an increasing complexity. Although for specific problems, especially when domain knowledge is available, methods such as Support Vector Machine (SVM) or Elastic Net (EN) cannot be beaten by more complex methods—especially under tight time and computational constraints. In these well specified settings, hand-crafted SVM kernel methods cannot be beaten by complex

methods. Therefore, in some well-defined domains, SVMs and related shallow methods can still be considered as efficient methods.

With increasing computational power and memory, more and more complex methods gain popularity. Some ML standard methods are cheap to evaluate, e.g., Decision Tree (DT), model complexity and performance increase: RF replaced simple trees and even more sophisticated methods such as XGBoost are considered state of the art.

The situation is getting worse when Deep Neural Networks (DNNs) are included: there is no limit for model complexity. As a consequence, HPT is developing very quickly to catch up with this exploding model complexity. New branches and extensions of existing HPT branches were proposed, e.g., Combined Algorithm Selection and Hyperparameter optimization (CASH), Neural Architecture Search (NAS), Automated Hyperparameter and Architecture Search (AutoHAS), and further "Auto-*" approaches (Thornton et al. 2013; Dong et al. 2020). The jungle of new ML and DL is accompanied by a plethora of HPT approaches and related software tools. Hutter et al. (2019) presents an overview of Automated Machine Learning (AutoML).

Although DL has been part of AI for a long time—first ideas go back to the 1940s—its break through happened in the early 2010s (McCulloch and Pitts 1943; Krizhevsky et al. 2012). Since 2012, Convolutional Neural Networks (CNNs) are the dominating approach in computer vision and image classification. In parallel, DL was adopted in several other domains, e.g., Natural Language Processing (NLP). DL methods outperformed standard ML methods such as SVMs in a wide range of applications (Chollet and Allaire 2018). Although finding good hyperparameters for shallow methods like SVM can be a challenging task, DL methods increased the difficulty significantly, because they explode the dimensionality of the hyperparameter space, Λ.

Therefore, it is worth looking at HPT strategies that were developed for DL. For example, Snoek et al. (2012) used the Canadian Institute for Advanced Research, 10 classes (CIFAR-10) data set, which consists of 60,000 32×32 color images in ten classes, for optimizing the hyperparameters of a CNNs. Bergstra et al. (2013) proposed a meta-modeling approach to support automated Hyperparameter Optimization (HPO), with the goal of providing practical tools that replace hand-tuning. They optimized a three-layer CNN. Eggensperger et al. (2013) collected a library of HPO benchmarks and evaluated three BO methods. Zoph et al. (2017) studied a new paradigm of designing CNN architectures and describe a scalable method to optimize these architectures on a data set of interest, for instance, the ImageNet classification data set.

The following example describes a typical approach of HPT in DL.

Example: Robust and Efficient Hyperparameter Optimization in DL

Falkner et al. (2018) optimized six hyperparameters that control the training procedure of a fully connected DNN (initial learning rate, batch size, dropout, exponential decay factor for learning rate) and the architecture (number of layers, units per layer)

Table 6.1 Elements of a typical HPT study: The hyperparameters and architecture choices for the fully connected networks as defined in Falkner et al. (2018)

Hyperparameter	Lower bound	Upper bound	Log-transform
Batch size	2^3	2^8	Yes
Dropout rate	0	0.5	No
Initial learning rate	$1e-6$	$1e-2$	Yes
Exponential decay factor	-0.185	0	No
# hidden layers	1	5	No
# units per layer	2^4	2^8	Yes

for six different data sets gathered from OpenML (Vanschoren et al. 2014), see Table 6.1.

The authors used a surrogate DNN as a substitute for training the networks directly. To build this surrogate, they sampled 10 000 random configurations for each data set, trained them for 50 epochs, and recorded the classification error after each epoch, and total training time. Two independent RF models were fitted to predict these two quantities as a function of the hyperparameter configuration used. Falkner et al. (2018) noted that Hyperband (HB) initially performed much better than the vanilla BO methods and achieved a roughly three-fold speedup over Random Search (RS).

Artificial toy functions were used in this study, and because BO does not work well on high-dimensional mixed continuous and categorical configuration spaces, they used a simple counting-ones problem to analyze this issue.

Tip: Handling mixed continuous, categorical, and combinatorial configuration spaces

Zaefferer et al. (2014) discussed these topics in great detail. How to implement BO for discrete (and continuous) optimization problems was analyzed in the seminal paper by Bartz-Beielstein and Zaefferer (2017). Furthermore, Zaefferer (2018) provides an in-depth treatment of this topic. In practice, `SPOT` can handle categorical and mixed variables as discussed in Sect. 4.5. Combinatorial problems, such as the optimization of permutations, strings, or graphs, can be treated by the R package `CEGO` (Zaefferer 2021).

Kedziora et al. (2020) analyzed what constitutes these systems and survey developments in HPO, e.g., multi-component models, Neural Network (NN) architecture search, automated feature engineering, meta-learning, multi-level ensembling, multiobjective evaluation, flexible user involvement, and principles of generalization, to name only a few.

Wistuba et al. (2019) described how complex DL architectures can be seen as combinations of a few elements, so-called *cells*, that are repeated to build the complete network. Zoph and Le (2016) were the first who proposed a cell-based approach, i.e., choices made about a NN architecture are the set of meta-operations and their arrangement within the cell. Another interesting example is function-preserving morphisms implemented by the Auto-Keras package to effectively traverse potential networks Jin et al. (2019).

NAS is discussed in (NAS Elsken et al. (2019)). Mazzawi et al. (2019) introduced a NAS framework to improve keyword spotting and spoken language identification models. Lindauer and Hutter (2020) describe AutoML for NAS.

Because optimizers can affect the DNN performance significantly, several tuning studies devoted to optimizers were published during the last years: Schneider et al. (2019) introduced a benchmarking framework called Deep Learning Optimizer Benchmark Suite (DeepOBS), which includes a wide range of realistic DL problems together with standardized procedures for evaluating optimizers. Schmidt et al. (2020) performed an extensive, standardized benchmark of fifteen particularly popular DL optimizers.

Menghani (2021) presented a survey of the core areas of efficiency in DL, e.g., spanning modeling techniques, infrastructure, and hardware accompanied by an experiment-based guide along with code for practitioners to optimize their model training and deployment.

Tunability, (see Definition 2.26) is an interesting concept that should be mentioned in the context of HPT (Probst et al. 2019a). The hope is that identifying *tunable* hyperparameters, i.e., ones that model performance is particularly sensitive to, will allow other settings to be ignored and results in a reduced hyperparameter search space, Λ. Unfortunately, tunability strongly depends on the choice of the data set, $(\mathcal{X}, \mathcal{Y})$, which makes a generalization of results very difficult.

Bischl et al. (2021a) provide an overview about HPO.

6.3 HPT Software Tools

The field of HPT software tools is under rapid development. Besides SPOT, which is discussed in this book, several other hyperparameter optimization software packages were developed. We will list packages that show a certain continuity and that hopefully will still be available in the near future.

The `irace` package implements the Iterated Race method, which is a generalization of the Iterated F-race method for the automatic configuration of optimization algorithms. Hyperparameters are tuned by finding the most appropriate settings for a given set of instances of an optimization problem. It builds upon the race package by Birattari et al. (2009) and it is implemented in R (López-Ibáñez et al. 2016).

The Iterative Optimization Heuristics profiler (IOHprofiler) is a benchmarking and profiling tool for optimization heuristics, composed of two main components (Doerr et al. 2018): The Iterative Optimization Heuristics analyzer (IOHanalyzer) provides

an interactive environment to evaluate algorithms' performance by various criteria, e.g., by means of the distribution on the fixed-target running time and the fixed-budget function values (Wang et al. 2022). The experimental platform, Iterative Optimization Heuristics experimenter (IOHexperimenter), is designed to ease the generation of performance data. Its logging functionalities allow to track the evolution of algorithm parameters, making the tool particularly useful for the analysis, comparison, and design of algorithms with (self-)adaptive hyperparameters. Balaprakash et al. (2018) presented DeepHyper, a PYTHON package that provides a common interface for the implementation and study of scalable hyperparameter search methods. Karmanov et al. (2018) created a "*Rosetta Stone*" of DL frameworks to allow data scientists to easily leverage their expertise from one framework to another. They provided a common setup for comparisons across GPUs (potentially CUDA versions and precision) and for comparisons across languages (Python, Julia, R). O'Malley et al. (2019) presented *Keras tuner*, a hyperparameter tuner for Keras with TensorFlow (TF) 2.0. Available tuners are RS and Hyperband. Mendoza et al. (2019) introduced *Auto-Net*, a system that automatically configures NN with SMAC by following the same AutoML approach as Auto-WEKA and Auto-sklearn. Zimmer et al. (2020) developed Auto-PyTorch, a framework for Automated Deep Learning (AutoDL) that uses Bayesian Optimization HyperBand (BOHB) as a back-end to optimize the full DL pipeline, including data pre-processing, network training techniques, and regularization methods. Mazzawi and Gonzalvo (2021) presented Google's Model Search, which is an open-source platform for finding optimal ML models based on TF. It does not focus on a specific domain.

Unfortunately, many of these software tools are results from research projects that are funded for a limited time span. When the project ends (and the developers successfully completed their Ph.D.s) the software package will not be maintained anymore. Despite the dynamics and volatility in this area, we do not want to shy away from giving an overview of the available software tools. Table 6.2 presents this overview, which should be regarded as an incomplete snapshot, but not as the whole picture of this field.

6.4 Summary and Discussion

Due to increased computational power, algorithm and model complexity grow into new regions. It is more and more important to understand the working mechanisms of complex neural networks. Putting the pieces together, it becomes clear that

1. there is a need for hyperparameter tuning,
2. surrogate optimization is an efficient approach, it can accelerate the search, and
3. mixed variable types (continuous, discrete) make hyperparameter tuning more difficult. Especially dependencies between different hyperparameters produce new challenges.

Table 6.2 Overview: HPT and HPO approaches

Software	Application	Method	Publication
AutoPyTorch	Fully automated DL (AutoDL)	BOHB	Zimmer et al. (2020)
Auto-Sklearn	Automated ML toolkit	BO, meta-learning and ensemble construction	Feurer et al. (2020)
Auto-WEKA	Search for the right WEKA ML algorithm and optimizes its hyperparameters	BO	Kotthoff et al. (2017)
BOHB	Distributed HB	BO and bandit-based methods	Falkner et al. (2018)
CAVE	Report generation	EDA, parameter importance analysis	Biedenkapp et al. (2018)
DEHB	Black-box optimization	HB, DE	Awad et al. (2021)
Google's model search	Build on TF, architecture search	multiple trainers, a search algorithm, a transfer learning algorithm. Database to store ML and DL models	Mazzawi and Gonzalvo (2021)
Hyperopt	Python library for serial and parallel optimization, can handle real-valued, discrete, and conditional dimensions	RS and TPEs	Bergstra et al. (2013), Koehrsen (2018)
IOHprofiler, IOHanalyzer, IOHexperimenter	analyze and visualize the empirical performance of IOHs, interactive plotting, statistical evaluation, report generation	R packages `Shiny`, `Plotly`, `Rcpp`	Doerr et al. (2018), Wang et al. (2022)
irace	Heuristics, automatic configuration of optimization and decision algorithms, appropriate settings of an algorithm given a set of instances of a problem	iterated racing	López-Ibáñez et al. (2016)
keras tuner	Hyperparameter tuner for keras/TF	RS, HB	O'Malley et al. (2019)
mlmachine	Uses Hyperopt as a foundation for performing experiments	BO	Koehrsen (2018)
Optuna	Software framework for ML	TPE, RS, grid search, CMA-ES	Akiba et al. (2019)
Ray-Tune	PyTorch, XGBoost, MXNet, and Keras and other frameworks	Wrapper around open-source optimization libraries such as HyperOpt, SigOpt, Dragonfly, and Facebook Ax	Liaw et al. (2018)
SMAC	Tool for algorithm configuration	BO, racing mechanism	Lindauer et al. (2022)
SPOT	Surrogate optimization	Various surrogates and optimizers, BO, RSM	Bartz-Beielstein et al. (2017)

To conclude this chapter, we would like to mention relevant criticism of HPT. Some authors even claimed that extensive HPT is not necessary at all. For example, Erickson et al. (2020) introduced a framework (AutoGluon-Tabular) that "requires only a single line of PYTHON to train highly accurate machine learning models on an unprocessed tabular data set such as a CSV file". AutoGluon-Tabular ensembles several models and stacks them in multiple layers. The authors claim that AutoGluon-Tabular outperforms AutoML platforms such as TPOT, H2O, AutoWEKA, auto-sklearn, AutoGluon, and Google AutoML Tables.

A highly recommendable study was performed by Choi et al. (2019), who presented a taxonomy of first-order optimization methods. Furthermore, Choi et al. (2019) demonstrated the sensitivity of optimizer comparisons to the hyperparameter tuning protocol: "optimizer rankings can be changed easily by modifying the hyperparameter tuning protocol." These results raise serious questions about the practical relevance of conclusions drawn from certain ways of empirical comparisons. They also claimed that tuning protocols often differ between works studying NN optimizers and works concerned with training NNs to solve specific problems.

Yu, Sciuto, Jaggi, Musat, and Salzmann (Yang and Shami) claimed that the evaluated state-of-the-art NAS algorithms do not surpass RS by a significant margin, and even perform worse in the Recurrent Neural Network (RNN) search space.

Balaji and Allen (2018) reported a multitude of issues when attempting to execute automatic ML frameworks. For example, regarding the random process, the authors state that "one common failure is in large multi-class classification tasks in which one of the classes lies entirely on one side of the train test split".

Li and Talwalkar (2019) stated that (i) better baselines that accurately quantify the performance gains of NAS methods, (ii) ablation studies (to learn about the NN by removing parts of it) that isolate the impact of individual NAS components, and (iii) reproducible results that engender confidence and foster scientific progress are necessary.

Liu (2018) remarks that "for most existent AutoML works, regardless of the number of layers of the outer-loop algorithms, the configuration of the outermost layer is definitely done by human experts". Human experts are shifted to a higher level, and are still in the loop. The lack of insights in current AutoML systems (Drozdal et al. 2020) goes so far that some users even prefer manual tuning as they believe they can learn more from this process (Hasebrook et al. 2022).

Taking this criticism seriously, we can conclude that transparency and interpretability of *both* the ML / DL method *and* the HPT process are mandatory. This conclusion becomes very important in safety-critical applications, e.g., security-critical infrastructures (drinking water), in medicine, or automated driving.

But in general, we can conclude, that HPT is a valuable, in some situations an even mandatory tool for understanding ML and DL methods. And, last but not least: HPT tools can help to gain trust in AI systems.