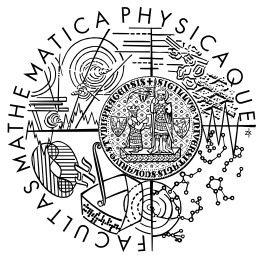

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jiří Krejčí

Efficient hyperparameter optimization

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the master thesis:	Mgr. Martin Pilát, Ph.D.
Study programme:	Computer Science - Artificial Intelligence
Study branch:	IUIP

Prague 2024

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

Dedication. It is nice to say thanks to supervisors, friends, family, book authors and food providers.

Title: Efficient hyperparameter optimization

Author: Jiří Krejčí

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Martin Pilát, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: Abstracts are an abstract form of art. Use the most precise, shortest sentences that state what problem the thesis addresses, how it is approached, pinpoint the exact result achieved, and describe the applications and significance of the results. Highlight anything novel that was discovered or improved by the thesis. Maximum length is 200 words, but try to fit into 120. Abstracts are often used for deciding if a reviewer will be suitable for the thesis; a well-written abstract thus increases the probability of getting a reviewer who will like the thesis.

Keywords: deep learning hyperparameter optimization (HPO) Bayesian optimization (BO) multi-fidelity

Contents

Introduction

In this thesis, we are addressing the problem of hyperparameter tuning of deep neural networks. The most common way to tune hyperparameters still to this day is by hand, by trial and error, relying on previous experience and various rules of thumb. It is very time-consuming for people and requires a great deal of expertise to do efficiently and well. With the steady increase in computing power, it is preferable to offload this task to computers. The right algorithm can do the task more thoroughly and without much human effort.

Even though this is an extensively studied problem, there still is no consensus on which method performs best. In this thesis, we want to expand on the knowledge of these algorithms and conduct experiments in the healthcare domain.

The main issue is that the function we optimize is expensive to evaluate and the range of possible input values is vast. One function evaluation, or training the network, can take hours or days. That is why the classical methods for hyperparameter optimization in machine learning such as grid search are not applicable to more complex deep learning architectures. Therefore, more sophisticated methods and algorithms were developed and are still being researched. The most important metric in this field is the efficiency of the search.

Hyperparameter optimization has roots in black-box optimization techniques. The machine learning models act as a black-box system to an extent. In deep learning, there is some potential to exploit the knowledge of the system, but only to a small degree. Many tools and optimization frameworks exist for hyperparameter tuning. Most of them are even open source, such as Optuna [1] or SMAC3 [2]. These tuning tools usually use Bayesian optimization techniques internally. Bayesian optimization is well suited for global optimization of black-box, expensive-to-evaluate functions.

In this thesis, we focus on low-budget search. We research the literature on multi-fidelity hyperparameter tuning and experiment with how to best spend a limited budget. We compare the HPO methods on some machine learning tasks in the domain of health care and others.

Chapter 1

Background and Literature Review

1.1 Machine learning fundamentals

Our goal is to optimize hyperparameters. But before we introduce the main topic, we need to specify the context. We are interested in supervised machine learning methods. Let $\mathscr{D}$ be a labeled dataset, consisting of examples generated independently from a data-generating distribution. Each example $(x^{(i)}, y^{i})$ consist of a feature vector $x^{(i)} \in \mathscr{X}$ and its label $y^{(i)} \in \mathscr{Y}$. Depending on what we want to predict, we distinguish classification and regression. In classification, the labels are finite-valued, while in regression the labels are real numbers. The goal of supervised learning is to train a model using the examples from the dataset $\mathscr{D}$ so that it generalizes well to data from the data-generating distribution.

We train machine learning algorithms by minimizing the loss function on the training data. Two loss functions are most commonly used. For regression, the mean square error of N examples is computed as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f(x^{(i)});\theta) - y^{(i)})^2$$

In the case of classification, we use the cross-entropy loss that measures the difference between two probability distributions. Let C be the number of classification classes and let $y^{(i)}$ be a distribution over all classes, which will often be just a one-hot encoding of the target class. Let the predictions of the model also form a distribution over the classes. Then the loss is calculated as follows when using mean reduction across examples

$$CE = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_c^{(i)} log(f(x^{(i)});\theta)_c)$$

For neural networks, the loss function is minimized using the gradient descent algorithm. Given a loss function $L(\theta)$ that we want to minimize with respect to

the model parameters, or weights, the gradient descent optimizes the function by repeatedly calculating the gradient $\nabla_\theta L(\theta)$ and performing an update to the weights in the opposite direction:

$$\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$$

Since the goal is for the model to generalize well, we need to estimate the generalization error during training. That is why we estimate it on data not used during training. We also want to verify the performance of the final model after we choose one. Not to introduce a bias to the estimate, this should also be done on unseen data. That is why it is a best practice to split the dataset into $\mathscr{D}_{train}, \mathscr{D}_{val}, \mathscr{D}_{test}$. As we will see, using a hyperparameter optimization tool introduces yet another optimization layer, meaning we should do a similar split even for the outer optimization of hyperparameters.

A neural network describes a computation. The basic unit is called a node and the nodes are arranged in an acyclic graph. Training of the network can be split into two parts — the forward pass and the backward pass. In the forward pass, the network produces an output from the input examples by iteratively going through the graph in a topological order. Then the loss is calculated. The gradients are computed using the backpropagation algorithm, which iterates through the nodes in reverse order and calculates partial derivatives of the loss with respect to the individual weights.

1.2 Hyperparameter optimization

Let A be a machine learning algorithm with hyperparameters $\lambda_1, \ldots, \lambda_n$ with domains $\Lambda_1, \ldots, \Lambda_n$. Let $\boldsymbol{\Lambda} = \Lambda_1 \times \cdot \times \Lambda_n$ denote its hyperparameter space. Hyperparameters can be continuous, integer-valued, or categorical. Also, we can have conditional hyperparameters. We say that hyperparameter λ_i is *conditional* on another hyperparameter λ_j, if λ_i is only active if hyperparameter λ_j takes values from a given set $V_i(j) \subset \Lambda_j$

For each hyperparameter configuration or setting $\lambda \in \boldsymbol{\Lambda}$, we denote A_λ the learning algorithm A using this hyperparameter setting. Let $\mathscr{L}(A_\Lambda, \mathscr{D}_{train}, \mathscr{D}_{valid})$ denote the validation loss of algorithm A_λ on data $\mathscr{D}_{valid}$ when trained on $\mathscr{D}_{train}$.

Definition 1 (Hyperparameter optimization problem)**.** *The hyperparameter optimization problem is to find hyperparameters λ^* that minimize the loss function*

$$\lambda^* = \arg\min_\lambda f(\lambda) = \arg\min_\lambda \mathscr{L}(A_\lambda, \mathscr{D}_{train}, \mathscr{D}_{valid}).$$

Several properties make the problem hard to solve.

- It is hard to obtain derivatives of the loss function with respect to hyperparameters and we will not use them to find the optimal solution. Such an optimization problem is called a black-box optimization in literature.
- Each function evaluation is expensive. Fully training a single deep neural network can take days.
- Each function evaluation may require a variable amount of time. For example, training larger models (e.g. more artificial neurons) takes more time to train. Therefore, the hyperparameter optimization algorithm should take training time into account.
- Observations are noisy. Repeated training may result in models that vary in performance since it is common to use random initialization of weights. The training process itself may not be deterministic as well. For example, if we use mini-batch shuffling.

On the other hand, we can leverage parallel computation to run multiple trials at the same time. One additional benefit of solving the optimization problem limited to deep neural networks is that we have access to intermediate results.

In this thesis, we assume that the general architecture of the neural network is already given and hyperparameters can change only smaller aspects, such as the number of neurons in a layer, or kernel size in a convolutional layer. For literature dealing with the more general problem, please refer to Neural Architecture Search (NAS).

For anyone interested in the process of hyperparameter tuning and how it might be done in practice, we recommend the Deep learning tuning playbook [3]. The authors give valuable insights into practical aspects of hyperparameter tuning that they have collected over more than ten years of working in deep learning. These insights are rarely documented. As the authors state in the text, they could not find any comprehensive attempt to explain how to get good results with deep learning. More importantly for this thesis, it gives us insight into how experts might do hyperparameter tuning. The text reveals that even today, advanced hyperparameter tuning tools are not the ultimate solution to the problem. Instead, they recommend how to use them smartly. They propose that there is still a human expert guiding the search, at least in the first, exploratory, phase.

1.3 Classic Hyperparameter Search Techniques

Before we get to algorithmic approaches, let us consider manual hyperparameter tuning. We cannot be surprised that people still tune hyperparameters manually.

There is no technical overhead or barrier. Also, in the process of hyperparameter tuning, we gain insight into the problem, which might allow us to improve our solution in ways that are not achievable just by hyperparameter tuning. Nevertheless, there are clear limits to manual tuning so let us dive into the automated approaches. The traditional algorithms for hyperparameter optimization are grid search and random search. These algorithms are simple and still widely used.

Grid Search Grid search performs an exhaustive search through a manually specified subset of the hyperparameter space. Grid search is best used when the number of hyperparameters is small, or the function evaluation is not that expensive. Its biggest drawback is that the number of configurations to evaluate grows exponentially with the number of hyperparameters. Therefore, it is best to determine which hyperparameters are the most important and limit the search only to this subset. If we did not do this, we would waste a lot of computational power on hyperparameter combinations, where only the unimportant hyperparameter changes, but the important ones stay the same. But how do we determine which hyperparameters are important and we need to tune them together? On the other hand, grid search is easily parallelizable. That is an enormous advantage since in real-world scenarios, it is not uncommon to have access to a computing cluster.

Random Search Random search is often used in the HPO literature as the baseline method for more advanced algorithms. In real-world optimization problems, random search often works better than the grid search. Bergstra et al. [4] compared random search to grid search and found that randomly chosen trials are more efficient for hyperparameter optimization than trials on a grid. It is possible to encounter a random search with 2X-budget as a baseline in some research papers. It is just a random search with two times the budget of other methods in comparison. As Li et al. [5] show, 2X-budget random search provides a strong baseline.

Quasi-random search If our budget is low then quasi-random search might be the better option. It works by generating a low-discrepancy sequence. Intuitively, a low-discrepancy sequence covers the whole domain evenly. Therefore, the search space is better covered even with a small number of samples. In the Deep learning tuning playbook, the authors recommend using quasi-random search over grid search and random search for the initial exploration of the hyperparameter space.

1.4 Bayesian optimization

So far, we have seen model-less approaches, where each trial is independent. This approach offers some advantages, like parallelization and simplicity of implementation, but it is quite inefficient. Information obtained from previous trials is not used in any way to guide the search. Bayesian optimization methods build and use an internal model of the learning algorithm's generalization performance. Bayesian optimization is widely covered in literature, we will use the work of Brochu et al. [6] and Frazier [7] for the definitions and description of the method.

The basic loop of a Bayesian optimization algorithm is simple. It uses the internal model to get a suggestion of the next hyperparameter configuration to try. Then it trains the neural network using the suggested configuration and uses the resulting performance metric to update the model. We repeat this process until we run out of budget or the neural network performs well enough.

In general, Bayesian optimization is a class of optimization methods focused on optimizing a real-valued objective function

$$\max_{x \in A} f(x).$$

Since we have defined the hyperparameter optimization problem as minimization of the loss function, we can assume that f is defined as $f(\lambda) = -\mathscr{L}(\lambda)$. Maximizing this function is equivalent to minimizing the original function. Also, not all hyperparameters are real-valued but we will address that later.

We assume that the objective is Lipschitz-continuous. That is, there exists some constant C such that for all $x_1, x_2 \in A : \|f(x_1) - f(x_2)\| \leq C\|x_1 - x_2\|$. The constant C may be unknown. The objective is commonly a black-box function, which in our case is true. We also assume that the search space is bounded in all dimensions.

The method got its name from the Bayes' theorem.

Practical BO of ML algorithms. They show how different kernels affect performance and describe algorithms that take into account the variable cost (duration) of learning algorithm experiments [8].

1.4.1 Gaussian process

"For continuous functions, Bayesian optimization typically works by assuming the unknown function was sampled from a Gaussian process and maintains a posterior distribution for this function as observations are made or, in our case, as the results of running learning algorithm experiments with different hyperparameters are observed (Practical Bayesian optimization 2012)" Good overview of Gaussian processes: [6].

1.4.2 Parzen-Tree Estimator

Algorithms for hyperparameter optimization [9] introduces TPE with Expected improvement.

1.5 Multifidelity techniques

Even though Bayesian optimization techniques are very sample-efficient, our budget might not allow for enough full training runs to find a good hyperparameter configuration. One option for dealing with such strict budget constraints is the multifidelity approach. A new hyperparameter λ_{fid} is introduced that allows us to trade off runtime for the reliability of the information we gain from evaluation. Low fidelity values mean that the evaluations are cheap but unreliable. Fidelity close to the upper limit implies that the returned values are close to true objective values achievable with full budget. What we are most interested in is finding λ_{fid} so that we use as few resources as possible and the rank correlation is mostly preserved between the models trained with low fidelity and the fully trained models. In other words, we want the low-fidelity models to reliably predict the ranking of the fully-trained models. In general, multifidelity HPO algorithms will start by evaluating cheap HPCs with low λ_{fid} values and then using the remaining budget to exploit the gathered data and train the most promising models with high λ_{fid}.

1.5.1 Early stopping

The simplest way to reduce the runtime of an algorithm is to stop the computation before it finishes. Swersky et al. [10] noticed that human experts have the ability to assess whether the model will eventually be useful early in the training and developed a method to leverage early stopping. They refer to this method as freeze-thaw Bayesian optimization, because it allows for pausing or aborting the training procedure when the model does not seem promising and resuming the training later if needed. This is combined with the Bayesian optimization framework for hyperparameter search and the authors propose a technique for estimating when to pause and resume training. The algorithm uses an information-theoretic criterion to determine which models to thaw.

Freeze-thaw method relies on the assumption that for many models the training loss roughly follows an exponential decay. Swersky et al. [10] developed a new kernel to serve as a prior characterizing the learning curves. This kernel is then used to forecast the final training loss and to provide these estimates to the Bayesian optimization. The kernel was successfully applied to matrix factorization and other problems, but it did not describe the learning curves of deep neural

networks well. The same idea was explored by Domhan et al. [11], but with a focus on deep neural networks. They developed a technique for extrapolation of learning curves based on a probabilistic model and used the model to terminate a training run when its performance is most likely going to be worse than the performance of the best model encountered so far. They modeled learning curves using eleven different model families and concluded that even though all of these models capture certain aspects of learning curves, no single model can describe all learning curves by itself. Therefore, they combined the models in a probabilistic framework. Their approach is agnostic to the hyperparameter optimizer and sped up the hyperparameter optimization approximately by a factor of two.

Hyperband

The Hyperband algorithm, developed by Li et al. [5], offers a completely different approach to dynamic resource allocation. It is a general approach with only a few assumptions. Authors originally developed it as an extension to speed up random search through early stopping.

The Hyperband internally uses the Successive Halving algorithm. Successive Halving works by uniformly allocating a budget to a set of hyperparameter configurations, then it throws out the worst half, and this process repeats until one configuration remains. The algorithm allocates exponentially more resources to more promising configurations, which usually speeds up the search considerably. In theory, the algorithm might never converge. For example, if the best configurations perform poorly in the beginning, then the algorithm discards them before they have the opportunity to converge. A practical downside of the algorithm is that we have to choose the number of configurations n in the first round manually. Therefore, we have to choose between large n, which means training a lot of hyperparameter configurations with a smaller training time and small n, resulting in the exploration of fewer hyperparameter configurations that are trained for longer on average.

The optimal choice of n depends mainly on how hard is it to distinguish similarly performing hyperparameter configurations from each other. We know that the intermediate losses are noisy, so we have to account for the uncertainty, which the authors of the paper call the envelope, as well. Ideally, we would wait until the envelopes do not overlap. We can observe that more resources are needed if the envelopes are wider, or if the terminal losses are closer together. The choice of n also places an upper bound on the execution time of a single configuration. The more configurations we want to evaluate, the less budget is allocated to the best configurations. Therefore, there might not be enough time for them to converge, and we might select a worse hyperparameter configuration as a result.

The Hyperband addresses the problem of selecting the optimal value of n by considering several possible values of n for a fixed budget B. Specifically, a successive halving algorithm is repeatedly called with different values of n. A larger value of n corresponds to more aggressive early-stopping since the same amount of computational resources is distributed between more hyperparameter configurations. Furthermore, by resetting the search, Hyperband is hedging against bad instantiations of the randomly sampled configurations and their initialization.

The drawback of Hyperband is that it does not scale well into larger budgets and random search starts to close the gap. Falkner et al. [12] noticed its deficiency and proposed a new algorithm BOHB to fix this. They combine Hyperband with Bayesian optimization in a way that their weaknesses are compensated by the other algorithm. Bayesian optimization needs some initial trials to gather enough data for the internal models, so it performs like a random search for a while. This is where the strong low-budget performance of Hyperband is used. On the other hand, well-fitted Bayesian optimization models provide better suggestions later in the tuning process.

Another algorithm extending the Hyperband is the DEHB developed by Awad et al. [13], which uses an evolutionary optimization method instead of Bayesian optimization. More specifically, the DE stands for Differential Evolution. The evolutionary approach provides some benefits over Bayesian optimization, such as better handling of discrete dimensions, better scaling into high dimensions, and conceptual simplicity enabling easy implementation.

The authors provided a lot of experiments in their paper, comparing DEHB to BOHB, random search, and other optimizers such as SMAC or Bayesian optimization with TPE surrogate. The benchmarks include NAS-Bench-101, NAS-HPO-Bench, or Reinforcement Learning Cartpole environment. The DEHB is much more efficient on some benchmarks while performing similarly to the BOHB, the next-best HPO optimizer in the experiments. The DEHB has very strong performance early with a low budget, and the performance does not fall off even for large budgets, which it often does for the BOHB.

1.5.2 Subsampling

The second possibility is to start the exploration with a fraction of the training data. The intuition behind this approach is that even a small subset of the training data will contain most of the information about the structure of the dataset for the model to learn. That should be enough to approximate the performance on the full dataset while training faster.

Klein et al. [14] developed a Bayesian optimization algorithm FABOLAS. The main idea of the algorithm is to introduce subset size as an additional parameter

for the Gaussian process to optimize using the acquisition function information gain per unit cost. The Gaussian process then learns to approximate the correlations between different subset size values, which allows it to efficiently use smaller subsets to accelerate the hyperparameter search. The authors performed experiments on the CIFAR10 and SVHN datasets with convolutional neural networks. FABOLAS found a good hyperparameter configuration more than 10 times faster than MTBO and the difference was even larger in comparison to the Hyperband. It is worth noting that after a good-performing model was found by all algorithms, the differences in test error were only minor.

A similar approach to BOHB was implemented by G. Zhu and R. Zhu [15]. They combine successive halving with progressively increasing dataset size and the number of training epochs. This way, the algorithm can explore even more configurations early on. The algorithm uses a Bayesian optimization with a surrogate model to suggest configurations for the successive halving, but the authors do not mention which surrogate model they used. To support the idea of using only a subset of the training data, the authors provide an experiment on the MNIST dataset, where they compared a LeNet trained on a full dataset versus trained only on 10% of the dataset. The results showed a difference of a few percentage points. The authors noted, that the main difference in chosen hyperparameters was in regularization hyperparameters. That is expected since training on a smaller dataset should require stronger regularization. Finally, they compared their algorithm to BOHB on CIFAR10 and CIFAR100 datasets. In both cases the new algorithm outperformed BOHB, especially will fewer resources used.

A method called DyHPO that is based on a novel Gaussian Process kernel was developed by Wistuba et al. [16]. They reviewed existing multifidelity approaches, including the Hyperband, BOHB, and DEHB. They have stated a conjecture that these gray-box methods suffer from a major issue — low-budget performances are not always a good indicator for the full budget performances. The authors argue as an example that a properly regularized network converges slower in the first few epochs, but typically outperforms a non-regularized network after full convergence. This problem is addressed by a GP kernel capable of capturing the similarity of two hyperparameter configurations even if the configurations are evaluated on different budgets.

Another area where DyHPO should gain efficiency compared to approaches like Hyperband is that instead of pre-allocating the budget, DyHPO dynamically adapts the allocation of budgets after every HPO step. Therefore, DyHPO invests only a small budget on unpromising configurations.

Some researchers explored the idea of a two-step hyperparameter optimization method — first optimize hyperparameters on a small subset of data and then optimize the best-performing models on the full dataset. This approach was

recently studied by Yu et al. [17] on a large dataset for aerosol activation emulator, containing almost 20 million examples. Their experiment is interesting because they use random search as an optimization algorithm and focus just on the dataset sizes, trying subsets as small as 0.00025 of the whole dataset (5000 examples). They have found that it does make sense to optimize hyperparameters on a small dataset first and that a lot of good models from the low-fidelity round perform well even on the full dataset. They were able to speed up the search 135 times while using just the simple and parallel random search, albeit on a single and very specific task.

1.6 Benchmarks

In this section, we review the literature comparing and benchmarking the approaches, which should further clarify which algorithm and when to use.

Li et al. [5] compare the Hyperband algorithm with three Bayesian optimization algorithms (SMAC, TPE, Spearmint) on CIFAR-10, rotated MNIST and SVHN datasets. They also include random search and 2x-random search as a baseline. The Hyperband consistently outperformed other algorithms at the beginning of the search. As the search progressed to spending the whole budget, the differences were only small between the methods.

FABOLAS [14] - FABOLAS>MTBO>Hyperband.

Chapter 2

Methodology

2.1 Algorithms

2.2 Datasets and models

2.3 Evaluation Metric

2.4 Experimental Setup

Chapter 3

Results and discussion

3.1 Discussion

Chapter 4

Related literature

Things that are not directly relevant, but related to HPO. Might not include this chapter in the final version. In this chapter, we mention other approaches in the literature that we did not use directly for solving the problem, but we think might be interesting for some readers.

4.1 Transfer learning

One advantage that an experienced practitioner will have over a classical HPO algorithm is that he will be good at generalizing and estimating good hyperparameter configurations across similar learning problems. The algorithm either depends on good bounds given by a user for efficient search, or it has to try a lot of configurations that do not perform well at all to find the bounds itself. The main idea of transfer learning is to use the experience from previous trials and similar problems in a new trial. This target function estimate should guide the search until the model is refined by new trials.

Collaborative hyperparameter tuning [18].

Efficient transfer learning method for automatic hyperparameter tuning [19].

Scalable hyperparameter transfer learning [20].

Pre-trained Gaussian processes for Bayesian optimization [21].

Conclusion

Bibliography

[1] Takuya Akiba et al. "Optuna: A Next-Generation Hyperparameter Optimization Framework". In: *The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 2623–2631.

[2] Marius Lindauer et al. "SMAC3: A Versatile Bayesian Optimization Package for Hyperparameter Optimization". In: *Journal of Machine Learning Research* 23.54 (2022), pp. 1–9. URL: http://jmlr.org/papers/v23/21-0888.html.

[3] Varun Godbole et al. *Deep Learning Tuning Playbook*. Version 1.0. 2023. URL: http://github.com/google-research/tuning_playbook.

[4] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization." In: *Journal of machine learning research* 13.2 (2012).

[5] Lisha Li et al. "Hyperband: A novel bandit-based approach to hyperparameter optimization". In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52.

[6] Eric Brochu, Vlad M Cora, and Nando De Freitas. "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv preprint arXiv:1012.2599* (2010).

[7] Peter I Frazier. "A tutorial on Bayesian optimization". In: *arXiv preprint arXiv:1807.02811* (2018).

[8] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms". In: *Advances in neural information processing systems* 25 (2012).

[9] James Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems* 24 (2011).

[10] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. "Freeze-thaw Bayesian optimization". In: *arXiv preprint arXiv:1406.3896* (2014).

[11] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves". In: *Twenty-fourth international joint conference on artificial intelligence*. 2015.

[12] Stefan Falkner, Aaron Klein, and Frank Hutter. "BOHB: Robust and efficient hyperparameter optimization at scale". In: *International conference on machine learning*. PMLR. 2018, pp. 1437–1446.

[13] Noor Awad, Neeratyoy Mallik, and Frank Hutter. "Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization". In: *arXiv preprint arXiv:2105.09821* (2021).

[14] Aaron Klein et al. "Fast bayesian optimization of machine learning hyperparameters on large datasets". In: *Artificial intelligence and statistics*. PMLR. 2017, pp. 528–536.

[15] Guanghui Zhu and Ruancheng Zhu. "Accelerating hyperparameter optimization of deep neural network via progressive multi-fidelity evaluation". In: *Advances in Knowledge Discovery and Data Mining: 24th Pacific-Asia Conference, PAKDD 2020, Singapore, May 11–14, 2020, Proceedings, Part I 24*. Springer. 2020, pp. 752–763.

[16] Martin Wistuba, Arlind Kadra, and Josif Grabocka. "Supervising the multi-fidelity race of hyperparameter configurations". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 13470–13484.

[17] Sungduk Yu et al. "Two-step hyperparameter optimization method: Accelerating hyperparameter search by using a fraction of a training dataset". In: *Artificial Intelligence for the Earth Systems* 3.1 (2024), e230013.

[18] Rémi Bardenet et al. "Collaborative hyperparameter tuning". In: *International conference on machine learning*. PMLR. 2013, pp. 199–207.

[19] Dani Yogatama and Gideon Mann. "Efficient transfer learning method for automatic hyperparameter tuning". In: *Artificial intelligence and statistics*. PMLR. 2014, pp. 1077–1085.

[20] Valerio Perrone et al. "Scalable hyperparameter transfer learning". In: *Advances in neural information processing systems* 31 (2018).

[21] Zi Wang et al. "Pre-trained Gaussian processes for Bayesian optimization". In: *arXiv preprint arXiv:2109.08215* (2021).