Hyper-Parameter Tuning using Bayesian Optimization

Pedro Gonçalo Mendes, 81046
Instituto Superior Técnico
Lisbon, Portugal
pedrogoncalomendes@tecnico.ulisboa.pt

***Abstract*—Hyper-parameter tuning is a critical step in machine learning training pipelines to ensure high quality models. This stage of the pipeline entails continuous deployments of training runs with different sets of parameters in search for the best performing ones. For this reason, this becomes a burdensome and expensive task.**

To address this issue, recent solutions exploit cheap, partially trained models, to extract information regarding complete models. These works can be divided on whether they exploit low fidelity or high-fidelity evaluations. Besides, the existing solutions in this area can also be classified depending on whether they are model-free or model-based approaches.

Several recent solutions rely on Bayesian Optimization (BO) techniques to select the hyper-parameters that maximize the quality to train a Machine Learning model. Moreover, several works propose the joint use of BO and low-fidelity observations (e.g., training with sub-sampled datasets or for short periods of time) to extrapolate good configurations to use when performing full training.

This work presents a collection of different systems found in the literature for hyper-parameter tuning. Furthermore, it gives more focus to works that leverage Bayesian Optimization methods to solve this non-trivial task. At last, it compares the different pros and cons of using BO in these systems for selecting the hyper-parameter, and it concludes with a critical analysis of the current literature, and the future work in this area, for example, how to extend these systems to solve other types of problems.

I. INTRODUCTION

Current Machine Learning (ML) models are getting larger and more complex. There are models with billions of hyper-parameters to optimize and, when using neural networks, an immensely vast space of alternative [1] viable architectures that exhibit complex trade-offs, e.g., for what concerns complexity and accuracy. Furthermore, the current trend is towards using increasingly large training datasets in order to improve the quality of the models As a result, even the testing of a single configuration (composed, e.g., of hyper-parameters) using these complex models and huge amounts of data can be extremely time consuming and expensive. Overall, the training process of modern ML models can require large amount of resources, take several hours, days, or even weeks, and lead to spending huge amounts of energy and emitting enormous quantities of CO_2 [1].

When using ML models, users face a very complex task: how to efficiently optimize the choice of the ML model's hyper-parameters while ensuring adequate Quality of Service (QoS) levels? Modern ML models have a large number of internal hyper-parameters (e.g., batch size considered in each training iteration or the frequency of synchronization among workers in the distributed training) that need to be tuned by the users during the training phase. This complex job is not independent of the chosen ML modeling technique, and the wrong selection of these hyper-parameters is likely to negatively impact the quality of the model. Several works [2]–[6] have addressed the problem of hyper-parameter tuning in order to increase the quality of the models or reduce the optimization time. Moreover, as shown in [3], it is important to jointly optimize the hyper-parameters and the resources to deploy the training in order to produce models with better quality. Other works [2], [4]–[6] investigated the use of low-fidelity evaluations (e.g., sub-sampling techniques, i.e., reduce the amount of data used to train the models), in order to decrease the training time.

This survey presents a set of works for hyper-parameter tuning. In particular, it is focused on systems that leverage Bayesian Optimization to solve the optimization problem and select the optimal hyper-parameters that, e.g., maximize the accuracy of the model. Moreover, this paper is structure as follows: § II introduces Bayesian optimization techniques; § II-A presents different modeling techniques used in the literature of BO; § III describes several state of the art systems for hyper-parameter optimization; and at last § IV concludes this paper, and presents some suggestions for future work in this area.

II. BAYESIAN OPTIMIZATION

Bayesian Optimization (BO) is a model-based techniques that aims to identify the optimum $\mathbf{x}^*$ of an unknown black-box function $f : \mathbb{X} \rightarrow \mathbb{R}$ and operates as follows [7]: (i) f is evaluated (i.e., tested or sampled) over N initial configurations, $\mathbf{x}_i$, selected at random so as to build an initial training set $\mathcal{S}$ composed of pairs $\langle \mathbf{x}_i, f(\mathbf{x}_i) \rangle$; (ii) $\mathcal{S}$ is used to train a black-box model (typically a Gaussian Process [8]) that serves as a predictor/estimator of the unknown function f; (iii) an *acquisition function* α is used to exploit the model's knowledge and uncertainty to determine which configuration to evaluate next by balancing exploitation of model's knowledge — recommending configurations that the model deems to be optimal — and exploratory behaviours — recommending configurations whose knowledge can reduce the model's uncertainty and enhance its accuracy; (iv) the process is iteratively repeated until a stopping condition is met, e.g., after a fixed budget is consumed or if the gains from further sampling are predicted to be marginal by the model (e.g., below a fixed threshold).

BO builds a model of the objective function using the tested points/configurations and exploits its knowledge and uncertainty to guide the exploration of the search space towards the optimum, i.e., it determines the points to evaluate based on the prior knowledge of f. After each evaluation, the model is updated with the new data using the Bayes' Theorem [9].

One of the most fundamental components of BO techniques is the *acquisition function* α that selects the next configurations to evaluate by balancing exploration and exploitation. The main goal of the acquisition function is to select the next configuration to test in order to accelerate the convergence towards the optimum. One of the first acquisition functions found in the literature is the Probability of Improvement (PI) [10] that aims at maximizing the probability of improvement of a configuration $\mathbf{x}$ over the current best optimum or incumbent $\mathbf{x}^*$. If we consider that the predictions follow a Normal distribution with mean $\mu(\mathbf{x})$ and standard deviation $\sigma(\mathbf{x})$ (which is normally the case when using BO and GPs), the PI is given by

$$\alpha_{PI}(\mathbf{x}) = P(f(\mathbf{x}) \geq f(\mathbf{x}^*)) = \Phi\left(\frac{\mu(x) - f(x^*)}{\sigma(x)}\right), \quad (1)$$

where $\Phi(\cdot)$ is a normal CDF. PI has a pure exploitative behaviour and it fails to evaluate configurations in regions with high uncertainty.

A very common acquisition function is the Expected Improvement (EI) [11], which exploits information of the model's uncertainty on an untested configuration $\mathbf{x}$ to estimate by how much $\mathbf{x}$ is expected to improve over the current incumbent.

$$\alpha_{EI}(\mathbf{x}) = \int \max(0, f(\mathbf{x}) - f(\mathbf{x}^*)) p(f(\mathbf{x})|\mathcal{S}) df(\mathbf{x}) \quad (2)$$

In the case of Normal distribution, the EI is simply

$$\alpha_{EI}(\mathbf{x}) = \begin{cases} (\mu(\mathbf{x}) - f(\mathbf{x}^*))\, \Phi(Z) + \sigma(\mathbf{x})\phi(\mathbf{x}), & \text{if } \sigma(\mathbf{x}) > 0 \\ 0, & \text{if } \sigma(\mathbf{x}) = 0 \end{cases} \quad (3)$$

where $Z = \frac{\mu(\mathbf{x}) - f(\mathbf{x}^*)}{\sigma(\mathbf{x})}$, $\phi(\cdot)$ and $\Phi(\cdot)$ are the PDF and CDF of the standard normal distribution, respectively. The constrained expected improvement (EIc) extends the EI in order to incorporate constraints in the optimization problem. For that, it multiplies the EI by the probability that $\mathbf{x}$ meets the constraints.

Lower confidence bound (LCB) [12] (or upper confidence bound (UCB)) selects a configuration to sample based on the mean and variance of the prediction of $\mathbf{x}$

$$\alpha_{LCB}(\mathbf{x}) = \mu(\mathbf{x}) - \kappa\sigma(\mathbf{x}), \quad (4)$$

where κ is a tunable parameter to balance exploitation vs exploration.

Entropy Search (ES) [13] is an alternative acquisition function that selects which configurations to evaluate by predicting the corresponding information gain (i.e., the entropy) on the optimum, rather than aiming to evaluate near the optimum (as in EI). It chooses the point that is predicted to yield the largest reduction of the entropy (i.e., decrease the uncertainty over the location of the optimum). ES (Eq. (5)) is based on the probability distribution $p_{opt}(\mathbf{x}|\mathcal{S})$, namely the likelihood that a configuration $\mathbf{x}$ belongs to the set of optimal configurations for f, given the current observations in $\mathcal{S}$. The information gain deriving from testing $\mathbf{x}$ is computed using the expected Kullback-Leibler divergence (relative entropy) between $p_{opt}(\cdot|\mathcal{S} \cup \{\mathbf{x}, y\})$ and the uniform distribution $u(\mathbf{x})$, with expectations taken over the model-predicted probability of obtaining measurement y at $\mathbf{x}$

$$\begin{aligned} \alpha_{ES}(\mathbf{x}) = \mathbb{E}_{p(y|\mathbf{x},\mathcal{S})} \Bigg[\int & p_{opt}(\mathbf{x'}|\mathcal{S} \cup \{\mathbf{x}, y\}) \\ & \cdot \log \frac{p_{opt}(\mathbf{x'}|\mathcal{S} \cup \{\mathbf{x}, y\})}{u(\mathbf{x'})} d\mathbf{x'} \Bigg]. \end{aligned} \quad (5)$$

It should be noticed that normally the configuration tested $\mathbf{x}$ may differ from the predicted incumbent $\mathbf{x}^*$. Moreover, ES does not have a closed-form expression and requires several non-trivial approximations. Its computation is extremely time consuming and demands large amount of resources. This problem is even exacerbated in larger search spaces, which require the use of some heuristic or optimizer [5], [14]–[16] to reduce the number of configurations to test in the acquisition function.

There are several works in the literature [17], [18] that extend the ES in order to incorporate constraints in the optimization problem. However, this constrained versions of ES-based acquisition functions, such as Predictive Entropy Search with Constraints (PESC) [17] and constrained Max-value Entropy Search (cMES) [18] use several other approximations (in particular, they make use of Bochner's theorem for a spectral approximation), which does not allow to use GPs with non-stationary kernels or any other modelling techniques (like decision trees). A simpler approach developed in TrimTuner [5] is to extend ES by factoring in the expected value of the distribution of the probability that the new predicted incumbent $\mathbf{x}^*$ comply with the constraints after testing $\mathbf{x}$. This way, the selection of the modeling technique is independent of the computation of the acquisition function. It should be noticed that this approach requires simulating the model (i.e., re-train the model with its predictions) in order to simulate the impact of testing $\mathbf{x}$ in the incumbent.

Swerzky et al. [19] were probably the first to propose an adaptation of the BO framework to take advantage of low-fidelity evaluations obtained using a training set of smaller dimensions. This idea was extended in Fabolas [2], which learns the lowest-fidelities that provide more knowledge about the optimum. These model-based solutions, such as MTBO or Fabolas, use transfer-learning techniques to extrapolate the quality of high-fidelity configurations based on low-fidelity observations. However, their practical effectiveness hinges on the actual ability of the model to capture the, often complex, relations between high and low quality configurations. Further, these methods typically rely on complex mathematical tools, which makes them computationally very expensive.

A related body of work [20]–[23] uses models (typically GPs) to predict the loss of a neural network as a function of both the hyper-parameters and the training iterations. Models are then used to extrapolate the full-training loss and cancel under-performing training runs.

Lastly, originally all these acquisition functions are greedy/myopic, i.e., they consider only one step ahead in the optimization process, which, as for all greedy heuristics, can lead to exploring the search space in sub-optimal ways. However, there are several state of the art systems [3], [24] that extend these myopic approaches with a look-ahead policy in order to simulate n steps ahead.

A. Modeling techniques

BO resorts to black-box techniques to build a model of the objective function or any additional constraints. The standard modeling technique used in the literature of BO is Gaussian Processes (GPs). However, any model that can offer additional information about its uncertainty can be used. Moreover, it is possible to use an ensemble of learners to have some measure about the uncertainty of the model.

GPs [8] represent the *de facto* standard modeling approach in BO, due to their analytical tractability and flexibility [2], [7]. Key to the tractability of GPs is that the outputted predictions follow, by construction, a Gaussian Distribution with known parameters. Moreover, it is the possibility to define specialized kernels that provides flexibility to GPs, by allowing them to imbue the model with domain-specific knowledge.

However, training GPs is notoriously an expensive process [7]. GPs have appealing features with small data sets (e.g., in the early stage of the optimization process, when only a handful of configurations have been tested), where their ability to incorporate prior knowledge via smooth kernel leads to good extrapolation. However, the training time of GPs grows cubically with the number of observations [25], which makes them inherently non-scalable. This efficiency issue can be avoided by using ensembles of scalable learners like decision trees (DTs), whose individual predictions can be used to fit a Gaussian distribution. DTs are known for their high efficiency, but they can not be directly used to replace GPs since, unlike GPs, DTs do not provide a measure of uncertainty of their prediction. It is possible to circumvent this problem by using an ensemble of DTs and injecting diversity among the various learners by generating their datasets drawing with replacement from the same dataset. Then, we can estimate the probability distribution for a prediction as a Gaussian with mean and standard deviation derived from the predictions of the ensemble. On the down side, ensemble methods typically generate diverse training sets for the individual learners by "hiding" different subsets of the original data set — which makes these methods inherently more data hungry than GPs.

A hybrid approach is proposed in HyperJump [26], where GPs are used during the initial stage of the exploration, where these can still be trained efficiently, and switch to an ensemble of decision trees when there have been gathered sufficient data to use the latter method effectively.

III. Hyper-Parameter Tuning using Bayesian Optimization

Existing hyper-parameter techniques can be coarsely classified along two dimensions: i) whether they use model-free or model-based approaches, and ii) whether they exploit solely high-fidelity (i.e., full-training) evaluations or also multi-fidelity ones.

HyperBand (HB) [4] is arguably the most prominent model-free approach at the moment. HB is based upon a randomized search procedure, called Successive Halving (SH) [27], which operates in stages of fixed "budget" (e.g., training time or training set size): at the end of stage i, the best performing $1/\eta\%$ configurations are selected to be evaluated in stage $i + 1$, where they will be allocated $\eta\times$ larger budget. By restarting the SH procedure over multiple, so called, brackets using different initial training budgets, HB provides theoretical guarantees of convergence to the optimum, incurring negligible computational overheads and outperforming state of the art optimizers (e.g., based on BO) that do not exploit low-fidelity observations. However, the random nature of HB also inherently limits its efficiency, as shown by recent model-based multi-fidelity approaches [2], [6]. Its random nature, combined with its SH-based search algorithm, makes it not only provably robust but also very competitive and lightweight when compared to several model-based approaches.

Moreover, the random nature of this solution provides theoretical guarantees of eventual convergence to the optimum and allows for efficient implementations; however, it also limits the convergence speed.

As for the model-based approaches, recent literature on hyper-parameter optimization has been dominated by BO methods, which relies on modeling techniques (e.g., GPs, Random Forests [28] or TPE [29]) to build a surrogate model of the function $f : \mathcal{X} \rightarrow \mathcal{R}$ to be optimized. The surrogate model is then used to guide the selection of the configurations to test via an *acquisition function* that tackles the exploration-exploitation dilemma.

BOHB [6] combines HB and BO to enhance its efficiency by combining the best of both techniques. It extends HB with BO to warm start it, i.e., to select (a fraction of) the configurations to include in a new HB bracket (based on EI). This way it speeds ups the convergence of HB by selecting configurations that are more promising to improve the current incumbent.

HyperJump [26] is a novel hyper-parameter optimization method that builds upon HB's robust search strategy and accelerates it via an innovative, model-based technique that aims at modeling the risk of shortcutting an HB bracket. The basic idea is to "jump" (i.e., skip either partially or entirely) some of HB's stages and, this way, reduce the number of configurations to be tested and accelerate the optimization process. The authors propose a novel risk modeling approach (called Expected Accuracy Reduction) to predict the risk of jumping that exploits the model's knowledge and uncertainty to quantify/estimate the expected reduction in the accuracy between the best configuration included in the stage after a jump and the best configuration discarded due to a jump.

There are also a vast set of works that rely only on BO to select the best hyper-parameters that solve different optimization problems (e.g., maximize the model's accuracy). Lynceus [3] is a cost-aware system that aims at jointly optimize the selection of the hyper-parameters to train an ML model and the (cloud) resources to rent to deploy the training phase. It minimizes the training cost/time while complying with several Quality of Service (QoS) constraints (e.g., a minimum accuracy of the trained model). To solve the optimization problem, it leverages BO with EIc/USD (i.e., the constraint Expected improvement normalized by the cost of testing a configuration). Lynceus also present a look-ahead [24] approach to foresee the impact of testing a configuration in future explorations (it presents a non-myopic method).

Although Lynceus aims at reducing the cost of training a model, it fails at exploiting low-fidelity evaluations. Fabolas [2] is one of the first systems for hyper-parameter tuning that resorts to these low-fidelity evaluations in order to reduce the time of testing a configuration. It aims at selecting the hyper-parameters of an ML model to maximize its accuracy but only tests configurations using sub-sampled datasets to reduce the amount of data used to train the model, and thus the training time. Fabolas presents a new acquisition function called information gain per unit cost that trades off the information that can be gain about the optimal configuration (on the full dataset) and the cost (i.e., execution time) of training the model in a given configuration and a sub-sampled dataset.

$$\alpha_F(\mathbf{x}, s) = \mathbb{E}_{p(y|\mathbf{x},s,\mathcal{S})}\left[\int p_{opt}^{s=1}(\mathbf{x'}|\mathcal{S} \cup \{\mathbf{x}, s, y\}) \cdot \log \frac{p_{opt}^{s=1}(\mathbf{x'}|\mathcal{S} \cup \{\mathbf{x}, s, y\})}{u(\mathbf{x'})} d\mathbf{x'}\right] \frac{1}{C(\mathbf{x}, s)}, \tag{6}$$

where $\mathbf{x}$ is a configuration, $s \in [0, 1]$ is the sub-sampling rate applied to the full dataset ($s = 1$ corresponds to the entire dataset), $p_{opt}^{s=1}$ is the distribution of the predicted optimum, C is the predicted cost/time of a configuration $\mathbf{x}$ using a sub-sampling rate s, and $\mathcal{S}$ is a set that contains the tested configurations.

Moreover, Fabolas builds two predictive models, one for accuracy and the other for cost, and it uses GPs with special kernels that capture how shifts of the dataset size affect both the quality (i.e., accuracy) and training efficiency (i.e., cost or execution time) of the target model. In other words, the kernels are designed to capture the expected impact on cost and accuracy deriving from the use of sub-sampling. Specifically, they use a kernel obtained by the inner product of a "general purpose" *Matérn* 5/2 kernel [30] and two custom kernels that encode, respectively, the expectation that accuracy and cost of an ML model grow normally with larger data-set sizes.

TrimTuner [5] extends both Lynceus and Fabolas to develop a cost and time aware system that optimizes both the cloud resources and the hyper-parameters of an ML model to maximize its quality/accuracy while ensuring QoS constraints (e.g., maximum training time). It proposes a new acquisition function that extends the one of Fabolas to exploit the use of low-fidelity evaluations while having into account additional user-defined QoS constraints.

$$\alpha_{TT}(\mathbf{x}, s) = \mathbb{E}_{p(a|\mathbf{x},s,\mathcal{S})}\left[\int p_{opt}^{s=1}(\mathbf{x'}|\mathcal{S}^A \cup \{\mathbf{x}, s, a\}) \cdot \log \frac{p_{opt}^{s=1}(\mathbf{x'}|\mathcal{S}^A \cup \{\mathbf{x}, s, a\})}{u(\mathbf{x'})} d\mathbf{x'}\right] \cdot \frac{1}{C(\mathbf{x}, s)} \cdot \mathbb{E}_{p(\mathbf{q},a|\mathbf{x},s,\mathcal{S})}\left[\prod^{q_i \in \mathbf{Q}} p(q_i(\mathbf{x}^*, s{=}1) \geq 0 | \mathcal{S} \cup \{\mathbf{x}, s, \mathbf{q}, a\}\right], \tag{7}$$

where $\mathbf{q}$ and a are the predicted values of the constraints and accuracy, $\mathbf{Q}$ are the defined constraints, $\mathbf{x}^*$ is the predicted current incumbent, and $\mathcal{S}^A$ is the set that contains the accuracy of tested configurations.

The computation of this acquisition function (and the ones based on ES) can be very expensive. Thus, TrimTuner presents two variants where it uses GPs (similar to Fabolas) or an ensemble of DTs. Moreover, the later can reduce the training time by 13$\times$ without reducing the quality of the recommended incumbents, achieving performance similar to EIc.

IV. Discussion, Conclusion, and Future Work

This work presented a survey on the automatic selection of hyper-parameters of ML models to maximize their accuracy and/or minimize their training time. These systems are normally divided into model-free or model-based approaches, and low-fidelities or high-fidelities evaluations. Moreover, to solve the optimization problem, the majority of systems in the literature resort to Bayesian Optimization that builds a model of the objective and constraints functions, and guides the optimization/exploration process by determining the configurations to test via an acquisition function that balances exploration vs. exploitation in order to converge as fast as possible to the optimum.

It was shown that although model-based approaches are computational and mathematically more challenging than model-free methods, the former can speed up and decrease the time spent evaluating configurations (i.e., training the model in a given set of parameters). Thanks to low-fidelity observations (e.g., sub-sampling techniques), it is possible to achieve even higher reductions of the training time without impact the quality of the final model. However, these systems (that exploit low-fidelity evaluations) normally resort to acquisition functions based on the Entropy Search and present large overheads to compute the acquisition function, which increases the optimization time. Nonetheless, several works tried to tackle this drawback and reduce the overhead by using cheaper modeling techniques, as an ensemble of decision trees.

As future work, it would be interesting to extend these BO methods for hyper-parameter tuning to solve other complex problems that users and data scientists face when using and deploying their models.

For example, before training the model, users also need to decide which ML approach to use (e.g., decision trees vs neural networks) and, if neural models are chosen (as it is increasingly the case), which neural architecture to adopt. This is an extremely complex task, given the large amount of different ML approaches existing in the literature and that the number of possible neural architectures is, arguably, even larger.

The optimization techniques present in this paper can represent a valuable starting point also to address the Neural Architectural Search (NAS) problem: in fact, the choice of the neural architecture can be seen as an instance of the problem of hyper-parameter optimization, provided that one can easily map the space of possible neural architecture to a generic hyper-parameter space. While conceptually straightforward, this mapping process is a practice far from being trivial given the high dimensionality of the neural architectural space that makes simplistic solutions (e.g., that simply consider any possible architecture) overly onerous to be used in practice.

Thus, it would be important to investigate how to extend the optimization techniques developed so far to incorporate recent NAS techniques that consider a continuous relaxation of the architecture representation, allowing an efficient search of the architecture using gradient descent [31]. These approaches are conceptually very elegant, as they allow for solving the NAS problem using the same optimization methods used to solve the problem of training a neural network (e.g., using back-propagation methods). On the other hand, several recent works [32], [33] have also highlighted that these differential approaches are likely to incur instability issues that can severely hinder the quality of the optimization's result. Thus, a promising future work is combining differential search methods, such as DARTS [31], with model-based optimization methods (e.g., BO) that exploit low-fidelity observations in order to obtain the best of the two approaches, namely quick convergence to optimal solutions as well as robustness and cost-efficiency.

References

[1] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *ACL*, 2019.

[2] A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter, "Fast bayesian optimization of machine learning hyperparamaters on large datasets," in *AISTATS*, vol. 54, 2017, pp. 528–536.

[3] M. Casimiro, D. Didona, P. Romano, L. Rodrigues, W. Zwanepoel, and D. Garlan, "Lynceus: Cost-efficient tuning and provisioning of data analytic jobs," in *ICDCS*, 2020.

[4] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *Journal of Machine Learning Research*, vol. 18, pp. 1–52, 2018.

[5] P. Mendes, M. Casimiro, P. Romano, and D. Garlan, "Trimtuner: Efficient optimization of machine learning jobs in the cloud via sub-sampling," in *MASCOTS*. IEEE, 2020.

[6] S. Falkner, A. Klein, and F. Hutter, "BOHB: Robust and efficient hyperparameter optimization at scale," in *ICML*, vol. 80, 2018, pp. 1437–1446.

[7] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," Tech. Rep. arXiv:1012.2599, 2010.

[8] M. A. Osborne, R. Garnett, and S. J. Roberts, "Gaussian processes for global optimization," in *LION*, 2009.

[9] J. Stone, *Bayes' Rule: A Tutorial Introduction to Bayesian Analysis*, 06 2013.

[10] H. J. Kushner, "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise," in *Journal of Basic Engineering*, vol. 86, no. 1. The American Society of Mechanical Engineers, 1964, pp. 97–106.

[11] J. Mockus, V. Tiesis, and A. Zilinskas, "The application of bayesian methods for seeking the extremum," in *Toward Global Optimization*, vol. 2. Elsevier, 1978, pp. 117–128.

[12] D. D. Cox and S. John, "A statistical method for global optimization," in *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, 1992, pp. 1241–1246.

[13] P. Henning and C. J. Schuler, "Entropy search for information-efficient global optimization," in *Journal of Machine Learning Research*, vol. 13, no. Jan, 2012, pp. 1809–1837.

[14] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, no. 1, p. 157–181, 1993.

[15] N. Hansen, *The CMA Evolution Strategy: A Comparing Review*. Springer Berlin Heidelberg, 2006, pp. 75–102.

[16] C. Andrieu and J. Thoms, "A tutorial on adaptive mcmc," *Statistics and computing*, vol. 18, no. 4, pp. 343–373, 2008.

[17] J. Hernández-Lobato, M. Gelbart, M. Hoffman, R. Adams, and Z. Ghahramani, "Predictive entropy search for bayesian optimization with unknown constraints," in *ICML*, 2015, pp. 1699–1707.

[18] V. Perrone, I. Shcherbatyi, R. Jenatton, C. Archambeau, and M. Seeger, "Constrained bayesian optimization with max-value entropy search," *ArXiv*, vol. abs/1910.07003, 2019.

[19] K. Swersky, J. Snoek, and R. P. Adams, "Multi-task bayesian optimization," in *NIPS*, vol. 2, 2013, pp. 2004–2012.

[20] ——, "Freeze-thaw bayesian optimization," *arXiv preprint arXiv:1406.3896*, 2014.

[21] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, 2015.

[22] Z. Dai, H. Yu, B. K. H. Low, and P. Jaillet, "Bayesian optimization meets Bayesian optimal stopping," in *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, 2019.

[23] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[24] R. Lam and K. Willcox, "Lookahead bayesian optimization with inequality constraints," in *Proceedings of the 30th Neural Information Processing Systems Conference*, 2017, pp. 1890–1900.

[25] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.

[26] P. Mendes, M. Casimiro, and P. Romano, "Hyperjump: Accelerating hyperband via risk modelling," 2021.

[27] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, 2016.

[28] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, 2001.

[29] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, vol. 24. Curran Associates, Inc., 2011, pp. 2546–2554.

[30] B. Matérn, *Spatial Variation*. Berlin, Germany: Springer-Verlag, 1986.

[31] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," 2019.

[32] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," 2019.

[33] K. Bi, L. Xie, X. Chen, L. Wei, and Q. Tian, "Gold-nas: Gradual, one-level, differentiable," 2020.