Frugal Optimization for Cost-related Hyperparameters

Qingyun Wu *, **Chi Wang***, **Silu Huang**
Microsoft Research
{Qingyun.Wu, Wang.Chi, Silu.Huang}@microsoft.com

Abstract

The increasing demand for democratizing machine learning algorithms calls for hyperparameter optimization (HPO) solutions at low cost. Many machine learning algorithms have hyperparameters which can cause a large variation in the training cost. But this effect is largely ignored in existing HPO methods, which are incapable to properly control cost during the optimization process. To address this problem, we develop a new cost-frugal HPO solution. The core of our solution is a simple but new randomized direct-search method, for which we provide theoretical guarantees on the convergence rate and the total cost incurred to achieve convergence. We provide strong empirical results in comparison with state-of-the-art HPO methods on large AutoML benchmarks.

Introduction

Machine learning algorithms usually involve a number of hyperparameters that have a large impact on model accuracy and need to be set appropriately for each task (Melis, Dyer, and Blunsom 2018). For practitioners to easily and confidently apply generic ML algorithms, methods that can automatically tune these hyperparameters at low cost are needed. It motivates research in efficient hyperparameter optimization (HPO) (Falkner, Klein, and Hutter 2018). HPO is generally considered as a black-box function optimization problem where evaluating the black-box function is expensive as training and validating a model can be time-consuming. Further, this evaluation cost can be directly affected by a subset of hyperparameters. For example, in gradient boosted trees, the variation of the number of trees and the depth per tree can result in a large variation on training and validation time. In such scenarios, sample efficiency cannot be directly translated to cost frugality. Unfortunately, there has not been a generic HPO formulation that considers the existence of such cost-related hyperparameters, as previous work is mostly focused on the case where the evaluation cost is constant or some special case of variable cost (detailed in Related Work Section). In this paper, we provide such a formulation to fill this gap and propose a cost-frugal solution.

*Equal contribution

Problem Formulation

Let $f(\mathbf{x})$ denote the validation loss of the concerned machine learning algorithm under the given training dataset[1] and hyperparameter configuration $\mathbf{x}$. Let $g(\mathbf{x})$ denote the evaluation cost[2] incurred when obtaining $f(\mathbf{x})$. In general, $g(\mathbf{x})$ is not constant in the hyperparameter configuration space $\mathcal{X}$. Instead, there may exist $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ such that $g(\mathbf{x}_1) \gg g(\mathbf{x}_2)$. The goal of an HPO algorithm π is to find $\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x})$. The total cost incurred during this loss optimization process is $G(\pi) = \sum_{i=1}^{K_\pi^*} g(\mathbf{x}_i)$, where K_π^* is the number of function evaluations involved when $\mathbf{x}^*$ is found by algorithm π. In this paper, we formulate the cost-frugal HPO problem as follows: the HPO algorithm π needs to find $\mathbf{x}^*$ while keeping its total cost $G(\pi)$ small. When cost is indeed constant with respect to $\mathbf{x}$, $G(\pi)$ naturally degenerates to the number of iterations for convergence times a constant, and thus π is naturally cost-frugal as long as it has a good convergence property. In the case where cost-related hyperparameters exist, this formulation enables a characterization of the HPO algorithm π's cost behavior.

Contribution

In this paper, we take a fresh and unique path of addressing this problem based on *randomized direct search*, and develop a frugal optimization method `CFO`. Our solution is designed toward both small iteration number before convergence and bounded cost per iteration, which lead to low total evaluation cost under mild conditions. Specifically, `CFO` is built upon our newly proposed randomized direct search method `FLOW`^{2}, which can have an approximately optimal total cost when minimizing loss. Our solution is backed by both theoretical and empirical studies.

On the theoretical side, we first prove that `FLOW`^{2} enjoys a convergence rate of $O(\frac{\sqrt{d}}{\sqrt{K}})$ even in the non-convex case under a common smoothness condition. This convergence result is of independent interest in the theory of derivative-free optimization. Then we prove that due to `FLOW`^{2}'s unique

[1] We assume the concerned ML algorithm, training datasets and validation methods are all fixed.

[2] Throughout this paper, evaluation cost refers to the computation/time needed for training and validating an ML algorithm with the given training and validation dataset.

update rule, when it is combined with a low-cost initialization, the cost in any iteration of FLOW^2 can be upper bounded under reasonable conditions, and the total cost for obtaining an ϵ-approximation of the loss is bounded by $O(d\epsilon^{-2})$ times the optimal configuration's cost by expectation. To the best of our knowledge, such theoretical bound on cost does not exist in any HPO literature.

On the empirical side, we perform extensive evaluations using a latest AutoML benchmark (Gijsbers et al. 2019) which contains large scale classification tasks. We also enrich it with datasets from a regression benchmark (Olson et al. 2017) to test regression tasks. Compared to existing random search algorithm and four variations of Bayesian optimization, CFO shows better anytime performance and better final performance in tuning a popular library XGBoost (Chen and Guestrin 2016) and deep neural networks on most of the tasks with a significant margin.

Related Work

The most dominating search strategy for HPO is Bayesian optimization (BO), which uses probabilistic models to approximate the blackbox function to optimize. Notably effective BO-based HPO methods include Gaussian process (GP) (Snoek, Larochelle, and Adams 2012), tree Parzen estimator (TPE) (Bergstra et al. 2011) and sequential model-based algorithm configuration method (SMAC) (Hutter, Hoos, and Leyton-Brown 2011). However, classical BO-based methods are mainly designed for minimizing the total number of function evaluations, which does not necessarily lead to low evaluation cost. Some recent work studies ways to control cost in HPO using multi-fidelity optimization. FABOLAS (Klein et al. 2017) introduces dataset size as an additional degree of freedom in Bayesian optimization. Hyperband (Li et al. 2017) and BOHB (Falkner, Klein, and Hutter 2018) try to reduce cost by allocating gradually increasing 'budgets' in the search process. The notion of budget can correspond to either sample size or the number of iterations for iterative training algorithms. These solutions assume the evaluation cost to be equal or similar for each fixed 'budget', which is not necessarily true when there exist cost-related hyperparameters. These solutions also require a predefined 'maximal budget' and assume the optimal configuration is found at the maximal budget. So the notion of budget is not suitable for modeling even a single cost-related hyperparameter whose optimal value is not necessarily at maximum, e.g., the number K in K-nearest-neighbor algorithm. The same is true for two other multi-fidelity methods BOCA (Kandasamy et al. 2017) and BHPT (Lu et al. 2019).

The only existing method for generic cost-related hyperparameters is Gaussian process with expected improvement per second (GPEIPS) (Snoek, Larochelle, and Adams 2012). It models the evaluation cost using another Gaussian process, and heuristically adds the estimated cost into the acquisition function. Although this method is applicable to generic cost-related hyperparameters, there is no theoretical guarantee on either the optimization of loss or cost.

Algorithm 1 FLOW^2

Inputs: Stepsize $\delta > 0$, initial point $\mathbf{x}_0$ which has a low cost, i.e., small $g(\mathbf{x}_0)$, and number of iterations K.
Initialization: Obtain $f(\mathbf{x}_0)$
for $k = 0, 1, 2, \ldots, K-1$ **do**
  Sample $\mathbf{u}_k$ uniformly at random from unit sphere $\mathbb{S}$
  if $f(\mathbf{x}_k + \delta\mathbf{u}_k) < f(\mathbf{x}_k)$ **then** $\mathbf{x}_{k+1} = \mathbf{x}_k + \delta\mathbf{u}_k$
  else if $f(\mathbf{x}_k - \delta\mathbf{u}_k) < f(\mathbf{x}_k)$ **then**
    $\mathbf{x}_{k+1} = \mathbf{x}_k - \delta\mathbf{u}_k$
  else $\mathbf{x}_{k+1} = \mathbf{x}_k$

Method

In this section we first present our proposed randomized direct search method FLOW^2; then we present theoretical guarantees about both convergence and the cost of FLOW^2; at last we present our cost-frugal HPO method CFO, which is built upon FLOW^2 and several practical adjustments.

FLOW^2

The proposed algorithm FLOW^2 is presented in Algorithm 1, taking the step size δ, the number of iterations K, and the initial value $\mathbf{x}_0$ as the input. We denote by $\mathbf{x}_k$ the incumbent configuration at iteration k. FLOW^2 proceeds as follows: at each iteration k, we sample a vector $\mathbf{u}_k$ uniformly at random from a $(d-1)$-unit sphere $\mathbb{S}$. Then we compare loss $f(\mathbf{x}_k + \delta\mathbf{u}_k)$ with $f(\mathbf{x}_k)$, if $f(\mathbf{x}_k + \delta\mathbf{u}_k) < f(\mathbf{x}_k)$, then $\mathbf{x}_{k+1}$ is updated as $\mathbf{x}_k + \delta\mathbf{u}_k$; otherwise we compare $f(\mathbf{x}_k - \delta\mathbf{u}_k)$ with $f(\mathbf{x}_k)$. If the loss is decreased, then $\mathbf{x}_{k+1}$ is updated as $\mathbf{x}_k - \delta\mathbf{u}_k$, otherwise $\mathbf{x}_{k+1}$ stays at $\mathbf{x}_k$.

FLOW^2 is a simple but carefully designed randomized direct search method. Intuitively speaking, when a low-cost initialization point is provided, FLOW^2 starts from a low-cost region and gradually moves towards low-loss region, attempting to avoid evaluations on high-cost and high-loss hyperparameters. Although the algorithm does not explicitly use the cost information except a requirement of a low-cost initial point, it does implicitly leverage relations between the configurations and their corresponding cost, and the loss and cost of each configuration. FLOW^2 has some very nice properties including bounded number of iterations and bounded cost per iteration, which together make FLOW^2 cost-frugal. We remark that existing randomized search methods fail to bound the total cost as we demonstrate analytically later in this section and empirically in the Experiment Section. In the following, we elaborate more on FLOW^2's cost frugality, along with the comparison with existing methods.

Insights about the frugality of FLOW^2. We first recognize that the exact analytic form of $g(\mathbf{x})$ is usually not completely present, mainly because hyperparameters can have complex interactions with the training and inference of an ML algorithm. $g(\mathbf{x})$ can be observed after evaluating $\mathbf{x}$ (i.e., after the training and validation of the ML algorithm using $\mathbf{x}$), and in some cases estimated after enough observations. It is highly non-trivial how to effectively incorporate the observed or estimated cost information into the search process without affecting the convergence of loss. For example, simply downweighting the priority of high-cost configurations (as

used in GPEIPS) may make an HPO algorithm spend a long time evaluating many cheap but high-loss configurations. In addition, from the engineering perspective, cost observations are not necessarily reliable or available depending on the runtime environment. Therefore, FLOW^2 is designed not to require the exact analytic form of cost function or estimation from observations. Instead, FLOW^2 specializes its new configuration proposal and incumbent configuration update rules while doing a randomized direct search. Perhaps surprisingly, our algorithm without explicit usage of any observed or estimated cost information outperforms previous work using that information. Our algorithm does implicitly leverage some properties of the cost function such as Lipschitz continuity, which are detailed in the cost analysis of FLOW^2 subsection. When FLOW^2's specialized update rules are combined with those properties, it can restrict its search in a subspace of the original search space where the cost of each point is not unnecessarily high, without violation of its convergence rate.

Here let us briefly explain how our update rules are specialized for the sake of cost frugality. Let $\mathbf{x}_k^{\text{best}}$ denote the currently best configuration (i.e., having the lowest loss) found in FLOW^2 at iteration k. Since at every iteration FLOW^2 first checks whether the proposed new points $\mathbf{x}_k \pm \delta \mathbf{u}_k$ can decrease loss before updating $\mathbf{x}_{k+1}$ to $\mathbf{x}_k \pm \delta \mathbf{u}_k$ or to $\mathbf{x}_k$, it guarantees at any iteration k, (*Property 1*) $\mathbf{x}_k = \mathbf{x}_k^{\text{best}}$ is always true; (*Property 2*) the next incumbent configuration $\mathbf{x}_{k+1}$ is always in a neighboring area of $\mathbf{x}_k$; (*Property 3*) except the initialization step, evaluations of new configurations are only invoked in line 5 or line 6 of Algorithm 1, which correspond to a cost of at most $g(\mathbf{x}_k + \delta \mathbf{u}_k) + g(\mathbf{x}_k - \delta \mathbf{u}_k)$. Property 1 and Property 2 can help us establish a bound on the difference between the cost of the new incumbent configuration $g(\mathbf{x}_{k+1})$ and the cost of the currently best point $g(\mathbf{x}_k^{\text{best}})$. If the starting point $\mathbf{x}_0$ is initialized at the low-cost region[3], we can further prove that $g(\mathbf{x}_{k+1})$ will not be too much larger than $g(\mathbf{x}^*)$. Combining the above conclusions with Property 3, we bound the cost incurred in each iteration of FLOW^2. Then with the convergence guarantee, we can finally bound the total cost incurred in FLOW^2 (detailed in Proposition 2 and Theorem 3).

Comparison with existing algorithms. Compared to our method, commonly used Bayesian optimization methods in HPO cannot guarantee Property 2 and 3 introduced before and thus can hardly have a theoretical guarantee on the total cost. Our method is closely related to directional direct search methods (Kolda, Lewis, and Torczon 2003) and zeroth-order optimization (Nesterov and Spokoiny 2017). Directional direct search methods can guarantee Property 1 and 2 but they usually cannot guarantee Property 3 and do not have a good convergence result. Our method inherits advantages of zeroth-order optimization (Nesterov and Spokoiny 2017) in terms of being able to use randomized function evaluations to approximate gradient information (directional derivative in our case) for general black-box functions, which is the key for our good convergence rate guarantee. However, zeroth-order optimization methods usually cannot guarantee Property 1. Because of the reasons above, it is difficult for both types of the aforementioned methods to establish a bound on the cost similar to FLOW^2. Our method also shares a similar spirit with hypergradient descent techniques (Maclaurin, Duvenaud, and Adams 2015; Pedregosa 2016) in the sense that we are all trying to use approximated gradient information (with respect to hyperparameters) to guide the search. However, the application of hypergradient descent techniques depends on how the training algorithm works while our method is applicable to general black-box functions.

[3]We call a region low-cost region if $g(\mathbf{x}) < g(\mathbf{x}^*)$ for all $\mathbf{x}$ in that region, and high-cost region otherwise.

Convergence of FLOW^2

Insights about convergence. From the convergence perspective, FLOW^2 is built upon the insight that if $f(\mathbf{x})$ is differentiable[4] and δ is small, $\frac{f(\mathbf{x}+\delta\mathbf{u})-f(\mathbf{x})}{\delta}$ can be considered as an approximation to the directional derivative of loss function on direction $\mathbf{u}$, i.e., $f'_{\mathbf{u}}(\mathbf{x})$. By moving toward the directions where the approximated directional derivative $\frac{f(\mathbf{x}+\delta\mathbf{u})-f(\mathbf{x})}{\delta} \approx f'_{\mathbf{u}}(\mathbf{x})$ is negative, it is likely that we can move toward regions that can decrease the loss. Following this intuition, we establish rigorous theoretical guarantees for the convergence of FLOW^2 in both non-convex and convex cases under a L-smoothness condition.

Condition 1 (L-smoothness)**.** *Differentiable function f is L-smooth if for some non-negative constant L, $\forall \mathbf{x}, \mathbf{y} \in \mathcal{X}$, $|f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^\intercal(\mathbf{y} - \mathbf{x}))| \leq \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$, where $\nabla f(\mathbf{x})$ denotes the gradient of f at $\mathbf{x}$.*

Proposition 1. *Under Condition 1, FLOW^2 guarantees $f(\mathbf{x}_k) - \mathbb{E}[f(\mathbf{x}_{k+1})|\mathbf{x}_k] \geq \delta \cdot c_d \|\nabla f(\mathbf{x}_k)\|_2 - \frac{L\delta^2}{2}$, in which $c_d = \frac{2\Gamma(\frac{d}{2})}{(d-1)\Gamma(\frac{d-1}{2})\sqrt{\pi}}$ and $\Gamma(\cdot)$ denotes the Gamma function.*

This proposition provides a lower bound on the expected decrease of loss for every iteration in FLOW^2, i.e., $f(\mathbf{x}_k) - \mathbb{E}[f(\mathbf{x}_{k+1})|\mathbf{x}_k]$, where expectation is taken over the randomly sampled directions $\mathbf{u}_k$.

Proof idea The main challenge in our proof lies in the fact that while $\mathbf{u}_k$ is sampled uniformly from the unit hypersphere, the update condition (Line 5 and 6, which are also designed with the purpose of cost control) filters certain directions, and complicates the computation of the expectation. The main idea of our proof is to partition the unit sphere $\mathbb{S}$ into different regions according to the value of the directional derivative. For the regions where the directional derivative along the sampled direction $\mathbf{u}_k$ has large absolute value, it can be shown that our moving direction is close to the gradient descent direction using the L-smoothness condition, which leads to large decrease in loss. We prove that even if the loss decrease for $\mathbf{u}$ in other regions is 0, the overall expectation of loss decrease is close to the expectation of absolute directional derivative over the unit sphere, which equals to $c_d \|f(\mathbf{x}_k)\|_2$.

[4]For non-differentiable functions we can use smoothing techniques, such as Gaussian smoothing (Nesterov and Spokoiny 2017) to make a close differentiable approximation of the original objective function.

In the following two theorems we characterize the rate of convergence to the global optimal point $\mathbf{x}^*$ in the convex case and to first-order stationary points in the non-convex case.

Theorem 1 (Convergence of FLOW^2 in the convex case)**.** *If f is convex and satisfies Condition 1, $\mathbb{E}[f(\mathbf{x}_K)] - f(\mathbf{x}^*) \leq e^{-\frac{\delta c_d K}{R}} r_0 + \frac{L\delta R}{2c_d}$, in which c_d is defined as in Proposition 1, $r_0 = f(\mathbf{x}_0) - f(\mathbf{x}^*)$ and $R = \max_{k\in[K]} \|\mathbf{x}_k - \mathbf{x}^*\|_2$. If $\delta \propto \frac{1}{\sqrt{K}}$, $\mathbb{E}[f(\mathbf{x}_K)] - f(\mathbf{x}^*) \leq e^{-\frac{c_d\sqrt{K}}{R}} r_0 + \frac{LR}{2c_d\sqrt{K}}$.*

Theorem 2 (Convergence of FLOW^2 in the non-convex case)**.** *Under Condition 1, $\min_{k\in[K]} \mathbb{E}[\|\nabla f(\mathbf{x}_k)\|_2] \leq \frac{r_0 + \frac{1}{2}LK\delta^2}{c_d(K-1)\delta}$, in which $\frac{1}{c_d} = O(\sqrt{d})$, and $r_0 = f(\mathbf{x}_0) - f(\mathbf{x}^*)$. By letting $\delta \propto \frac{1}{\sqrt{K}}$, $\min_{k\in[K]} \mathbb{E}[\|\nabla f(\mathbf{x}_k)\|_2] = O(\frac{\sqrt{d}}{\sqrt{K}})$.*

The proof of the theorems above is based on Proposition 1. It is obvious that when the cost is constant, a good convergence of loss directly translates to a good bound on the total cost. However this is not necessarily true when cost-related hyperparameters exist, in which case a naive upper bound for $G(\pi)$ can be as large as $K^*_\pi \max_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x})$, recalling that K^*_π denotes the number of iterations used by π to find $\mathbf{x}^*$.

Cost Analysis of FLOW^2

In this section, we provide a rigorous analysis of the cost behavior of FLOW^2.

Condition 2 (Lipschitz continuity of cost function $g(\mathbf{x})$)**.** *$\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, $|g(\mathbf{x}_1) - g(\mathbf{x}_2)| \leq U \times z(\mathbf{x}_1 - \mathbf{x}_2)$, in which U is the Lipschitz constant, and $z(\mathbf{x}_1 - \mathbf{x}_2)$ is a particular distance function of $\mathbf{x}_1 - \mathbf{x}_2$. For example $z(\cdot)$ can be the Euclidean norm on the cost-related subset of the dimensions.*

Condition 2 recognizes the existence of a certain degree of continuity in the cost function. Using the gradient boosted trees example, let $\mathbf{x} = (x_1, x_2, x_3)$, where the three coordinates represent the tree number, max tree depth and learning rate respectively. The assumption implies that the difference between $g((50, 10, l_1))$ and $g((51, 11, l_2))$, where l_1 and l_2 are two settings of the learning rate, should not be too large.

Using the notations defined in Condition 2, we define $D \coloneqq U \times \max_{\mathbf{u}\in\mathbb{S}} z(\delta\mathbf{u})$, which is intuitively the largest distance between the points in two consecutive iterations of FLOW^2 considering the fact that the new point is sampled from the $(d-1)$-unit sphere surrounding the incumbent point. We denote by $\tilde{\mathbf{x}}^*$ a locally optimal point of f.

Condition 3 (Local monotonicity between cost and loss)**.** *$\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$, if $2D + g(\tilde{\mathbf{x}}^*) \geq g(\mathbf{x}_1) > g(\mathbf{x}_2) \geq g(\tilde{\mathbf{x}}^*)$, then $f(\mathbf{x}_1) \geq f(\mathbf{x}_2)$.*

Condition 3 states that when the cost surpasses a locally optimal point $\tilde{\mathbf{x}}^*$'s cost, i.e., $g(\mathbf{x}) \geq g(\tilde{\mathbf{x}}^*)$, with the increase of the evaluation cost in a local range, the loss does not decrease. Intuitively, for most ML models, when the model's complexity[5] is increased beyond a suitable size, the model's performance would not improve with the increase on the model's complexity due to overfitting.

[5]Model complexity is usually positively correlated with evaluation cost.

Proposition 2 (Bounded cost change in FLOW^2)**.** *If Condition 2 is true, then $g(\mathbf{x}_{k+1}) \leq g(\mathbf{x}_k) + D$, $\forall k$.*

Proposition 3 (Bounded cost for any function evaluation of FLOW^2)**.** *Under Condition 2 and 3, if $g(\mathbf{x}_0) < g(\tilde{\mathbf{x}}^*)$, then $g(\mathbf{x}_k) \leq g(\tilde{\mathbf{x}}^*) + D$, $\forall k$.*

Proposition 3 asserts that the cost of each evaluation is always within a constant away from the evaluation cost of the locally optimal point. The high-level idea is that FLOW^2 will only move when there is a decrease in the validation loss and thus the search procedure would not use much more than the locally optimal point's evaluation cost once it enters the locally monotonic area defined in Condition 3.

Let $\tilde{G}(\text{FLOW}^2)$ denotes the expected total evaluation cost for FLOW^2 to approach a first-order stationary point $f(\tilde{\mathbf{x}}^*)$ within distance ϵ, and K^* as the expected number of iterations taken by FLOW^2 until convergence. According to Theorem 2, $K^* = O(\frac{d}{\epsilon^2})$.

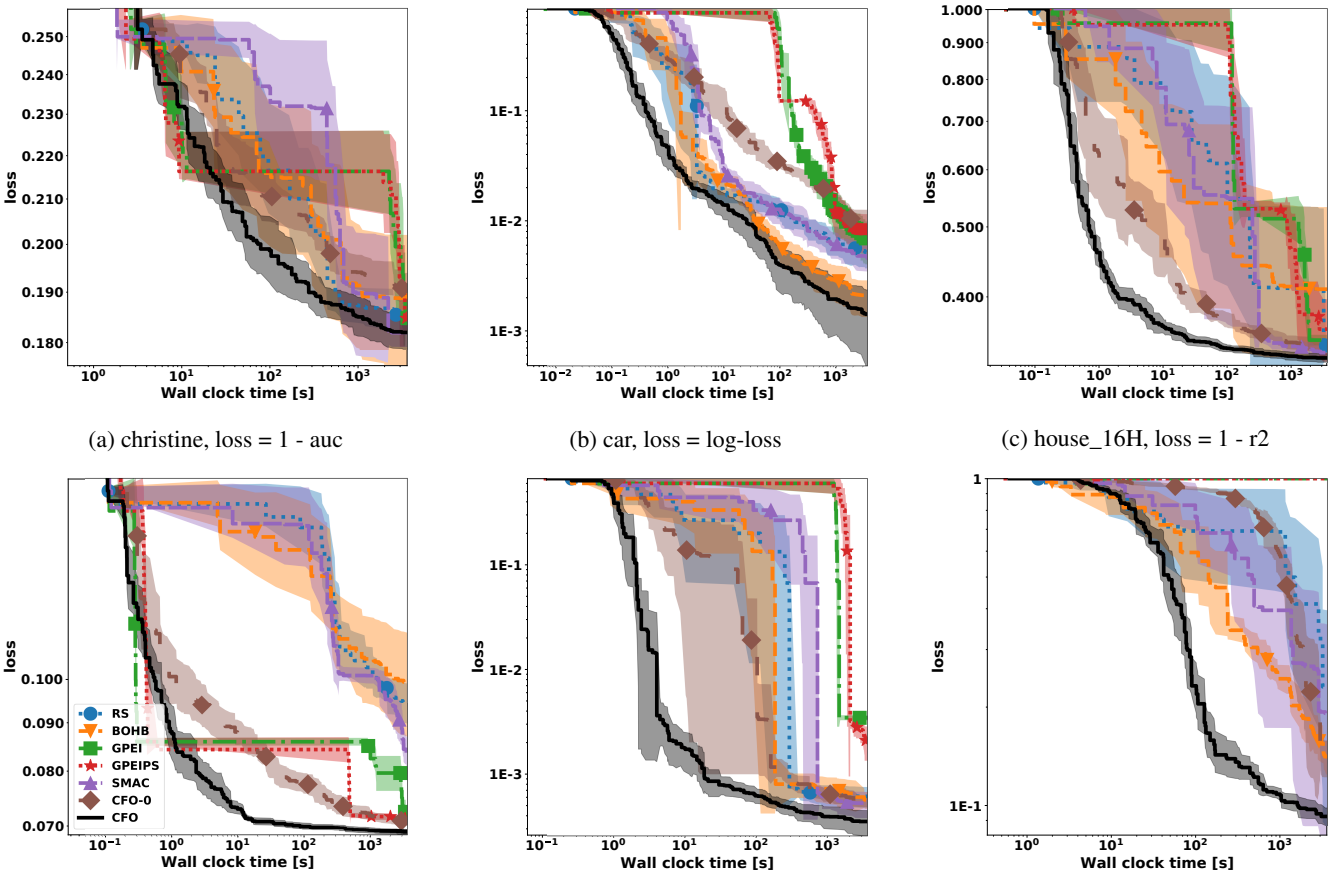
Theorem 3 (Expected total evaluation cost of FLOW^2)**.** *Under Condition 2 and Condition 3, if $K^* \leq \lceil \frac{\gamma}{D} \rceil$, $\tilde{G}(\text{FLOW}^2) \leq K^*(g(\tilde{\mathbf{x}}^*) + g(\mathbf{x}_0)) + 2K^*D$; else, $\tilde{G}(\text{FLOW}^2) \leq 2K^* g(\tilde{\mathbf{x}}^*) + 4K^*D - (\frac{\gamma}{D} - 1)\gamma$, in which $\gamma = g(\tilde{\mathbf{x}}^*) - g(\mathbf{x}_0) > 0$.*

Theorem 3 shows that the total evaluation cost of FLOW^2 depends on the number of iterations K^*, the maximal change of cost between two iterations D, and the evaluation cost gap γ between the initial point $\mathbf{x}_0$ and $\tilde{\mathbf{x}}^*$. From this result we can see that as long as the initial point is a low-cost point, i.e., $\gamma > 0$, and the step size is not too large such that $D \leq g(\mathbf{x}^*)$, the evaluation cost is always bounded by $\tilde{G}(\text{FLOW}^2) \leq 4K^* \cdot (g(\tilde{\mathbf{x}}^*) + g(\mathbf{x}_0) + D) = O(d\epsilon^{-2}) g(\tilde{\mathbf{x}}^*)$. Notice that $g(\tilde{\mathbf{x}}^*)$ is the minimal cost to spend on evaluating the locally optimal point $\tilde{\mathbf{x}}^*$. Our result suggests that the total cost for obtaining an ϵ-approximation of the loss is bounded by $O(d\epsilon^{-2})$ times that minimal cost by expectation. When g is a constant, our result degenerates to the bound on the number of iterations. We have not seen cost bounds of similar generality in existing work.

Remark 1. *To the best of our knowledge, the only theoretical analysis for HPO problems that considers cost appears in Hyperband (Li et al. 2017). They derived a theoretical bound on the loss with respect to the input budget. However, as introduced in the Related Work Section, their notion of 'budget' is not suitable for modeling generic cost-related hyperparameters.*

Remark 2. *Theorem 3 holds as long as Lipschitz continuity (Condition 2) and local monotonicity (Condition 3) are satisfied. It does not rely on the smoothness condition. So the cost analysis has its value independent of the convergence analysis. This general bound can be further reduced when the computational complexity of the ML algorithm with respect to the hyperparameters is partially known (in $\Theta(\cdot)$ notation instead of the exact analytic form).*

Due to the space limit, all proof details and how the bound on the total cost can be further reduced are deferred to the technical report of this paper (Wu, Wang, and Huang 2020).



(a) christine, loss = 1 - auc (b) car, loss = log-loss (c) house_16H, loss = 1 - r2

(d) adult, loss = 1 - auc (e) shuttle, loss = log-loss (f) poker, loss = 1 - r2

Figure 1: Performance over time. Lines correspond to mean loss over 10 folds, and shades correspond to 95% confidence intervals

Practical Adjustments

Despite the theoretical guarantee on its good convergence rate and expected total cost, vanilla FLOW^2 presented in Algorithm 1 is not readily applicable for HPO problems because (1) the possibility of getting stuck into local optima; (2) stepsize is needed as a hyperparameter of FLOW^2; and (3) the existence of discrete hyperparameters. Fortunately, those limitations can be effectively addressed using commonly used practical techniques in optimization. By adopting the practical adjustments listed below, we turn FLOW^2 into an off-the-shelf HPO solution, which is named as CFO (short for Cost-Frugal Optimization) and presented in Algorithm 2. It is also worth mentioning that in our empirical evaluations, we used the same practical adjustments on an existing zeroth-order optimization method in order to verify the unique advantages of FLOW^2 in terms of frugality.

Randomized restart of FLOW^2. Similar to most of the other local search algorithms, FLOW^2 may suffer from getting trapped in a local optimum. One common solution to relieve this pain is to restart the algorithm when no progress is observed (Martí, Moreno-Vega, and Duarte 2010; Zabinsky, Bulger, and Khompatraporn 2010; György and Kocsis 2011). Following the same spirit, we restart CFO from a randomized starting point when no progress is made in it. Specifically, in our work, the 'no progress' signal is determined by the number of consecutive no improvement interactions and the value of stepsize. The randomized starting point is generated by adding a Gaussian noise $\mathbf{g}$ to the original initial point $\mathbf{x}_0$.

Dynamic adjustments of stepsize δ. To achieve the convergence rate proved before, the stepsize of FLOW^2, i.e., δ needs to be set as a constant that is proportionally to $1/\sqrt{K*}$, which is difficult to be specified beforehand. Adaptive stepsize is prevalent in iterative optimization and search algorithms (Boyd, Boyd, and Vandenberghe 2004). In our work, we propose a self-adjustable stepsize rule, which shares the same spirit with the adaptive rule in (Konnov 2018). The stepsize is initially δ_{init}, which we set to be $\sqrt{d}$. It will be decreased when the number of consecutively no improvement iterations is larger than 2^{d-1}. Specifically, δ is discounted by a factor of $\frac{1}{\sqrt{\eta}}$, in which the reduction ratio $\eta > 1$ is intuitively the ratio between the total number of iterations taken since the last restart and the total number of iterations taken to find the best configuration since the last restart. By doing so, the stepsize reduction ratio η does not need to be pre-specified but is self-adjustable to the progress made by

Algorithm 2 CFO

Inputs: 1. Feasible search space $\mathcal{X}$, the dimensionality of which is d. 2. Initial low-cost configuration $\mathbf{x}_0$.
Initialization: Set initial value $\mathbf{x} = \mathbf{x}_0$ (and get $f(\mathbf{x})$), $\delta = \delta_{init}$, $k = k' = n = r = 0$, $l^{\text{r-best}} = +\text{inf}$
while Budget allows **do**
  Sample $\mathbf{u}$ uniformly at random from $\mathbb{S}$
  $\mathbf{x}^+ \leftarrow \text{Proj}_{\mathcal{X}}(\mathbf{x} + \delta\mathbf{u})$, $\mathbf{x}^- \leftarrow \text{Proj}_{\mathcal{X}}(\mathbf{x} - \delta\mathbf{u})$
  if $f(\mathbf{x}^+) < f(\mathbf{x})$ **then** $\mathbf{x} \leftarrow \mathbf{x}^+$
  else if $f(\mathbf{x}^-) < f(\mathbf{x})$ **then** $\mathbf{x} \leftarrow \mathbf{x}^-$
  else $n \leftarrow n + 1$
  if $f(\mathbf{x}) < l^{\text{r-best}}$ **then** $l^{\text{r-best}} \leftarrow f(\mathbf{x})$ and $k' \leftarrow k$
  $k \leftarrow k + 1$
  if $n = 2^{d-1}$ **then**
    $n \leftarrow 0$, $\delta \leftarrow \delta \frac{1}{\sqrt{\eta}}$, in which $\eta = \frac{k}{k'}$
    if $\delta \leq \delta_{\text{lower}}$ **then**
      $k \leftarrow 0$, $l^{\text{r-best}} \leftarrow +\text{inf}$
      Reset $\mathbf{x} \leftarrow \mathbf{x}_0 + \mathbf{g}$, where $\mathbf{g} \sim N(0, \mathbf{I})$
      $r \leftarrow r + 1$ and $\delta \leftarrow r + \delta_{init}$

the search. In order to prevent δ from becoming too small, we also impose a lower bound on it and stop decreasing δ once it reaches the lower bound δ_{lower}, which is also designed in a self-adjustable manner.

Projection of the proposed configuration. In practice, the newly proposed configuration $\mathbf{x} \pm \mathbf{u}$ is not necessarily in the feasible hyperparameter space $\mathcal{X}$, especially when discrete hyperparameters exist. In such scenarios, we use a projection function $\text{Proj}_{\mathcal{X}}(\cdot)$ to map it to the feasible space $\mathcal{X}$.

We provide the detailed justifications on the design rationale of CFO in (Wu, Wang, and Huang 2020).

Discussions

(1) No tuning needed. We realized the practical adjustments in CFO in a way that is self-adjustable and does not require on any tuning. *(2) Low-cost initialization.* The low-cost initialization is fairly easy to perform in practice because we do not have any requirement on the performance (in terms of loss) of it. *(3) Parallelization.* Our method is easy to be parallelized. When extra resource is available, instead of doing purely sequential random restarts, we can start multiple FLOW^2 search threads with different initial points and run them in parallel. *(4) Categorical hyperparameters.* Our current method FLOW^2 is primarily designed for the optimization of numerical hyperparameters with solid theoretical guarantees. In practice, it is possible to extend it to handle categorical hyperparameters. For example, we can encode categorical choices as integers. But instead of using a fixed mapping between the integer encoding and the categorical choices, we randomly reassign the categorical choices when the projected integer for a categorical dimension changes.

CFO is available in an open-source AutoML library FLAML[6] with all the extensions discussed.

[6]https://github.com/microsoft/FLAML/tree/main/flaml/tune

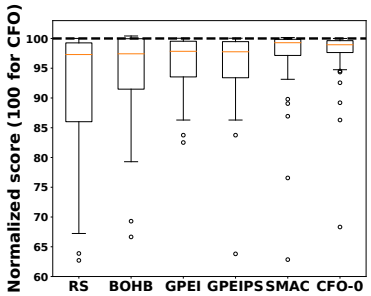
Experiment

We perform an extensive experimental study using a latest open source AutoML benchmark (Gijsbers et al. 2019), which includes 39 classification tasks. We enriched it with 14 regression tasks[7] from PMLB (Olson et al. 2017). All the datasets are available on OpenML. Each task consists of a dataset in 10 folds, and a metric to optimize: Roc-auc for binary tasks, log-loss for multi-class tasks, and r2 score for regression tasks. We include 5 representative HPO methods as baselines, including random search (RS) (Bergstra and Bengio 2012), Bayesian optimization with Gaussian Process and expected improvement (GPEI) and expected improvement per second (GPEIPS) as acquisition functions respectively (Snoek, Larochelle, and Adams 2012), SMAC (Hutter, Hoos, and Leyton-Brown 2011), and BOHB (Falkner, Klein, and Hutter 2018). The latter four are all based on Bayesian optimization. Among them, BOHB was shown to be a state of the art multi-fidelity method. We consider the training sample size to be the resource dimension required in BOHB. GPEIPS considers cost by building a probabilistic model about the configuration cost and using expected improvement per second as the acquisition function. In addition to these existing HPO methods, we also include an additional method CFO-0, which uses the same framework as CFO but replaces FLOW^2 with the zeroth-order optimization method ZOGD (Nesterov and Spokoiny 2017). Notice that like FLOW^2, ZOGD is not readily applicable to the HPO problem, while the CFO framework permits ZOGD to be used as an alternative local search method. The comparison between CFO-0 and CFO would allow us to evaluate the contribution of FLOW^2 in controlling the cost in CFO. All the methods start from the same initial point as the one used in CFO.

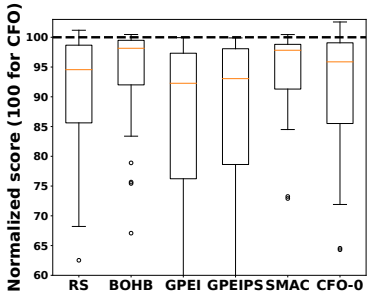
We compare their performance in tuning 9 hyperparameters for XGBoost, which is one of the most commonly used libraries in many machine learning competitions and applications. In addition to XGBoost, we also evaluated all the methods on deep neural networks. The experiment setup and comparison conclusion are similar to those in the XGBoost experiment. Due to space limit, we the results on deep neural networks in (Wu, Wang, and Huang 2020).

Performance curve. To investigate the effectiveness of CFO's cost control, we visualize the performance curve over an one-hour wall clock time period. We include the performance curves in tuning XGBoost on 6 datasets in Figure 1 and put the rest in the appendix. These 6 datasets represent a diverse group: The two rows in Figure 1 include three small datasets and large datasets respectively. In each row, the three datasets are for binary classification, multi-class classification, and regression task respectively. The curves show that overall it takes RS and classical BO-based methods much longer time to reach low loss, because they are prone to trying unnecessarily expensive configurations. In our experiments, GPEIPS outperforms GPEI in some cases (for example, *adult*) but tends to underperform GPEI on small datasets (for example, *car*) probably due to its penalization on good but expensive configurations even when budget is

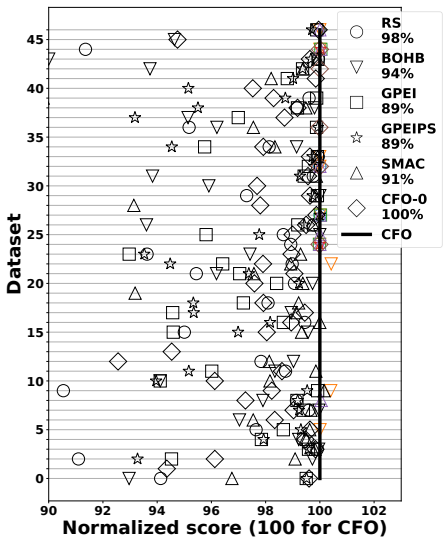
[7]Among the 120 regression datasets in PMLB, we selected the ones whose # of instances are larger than 10K.



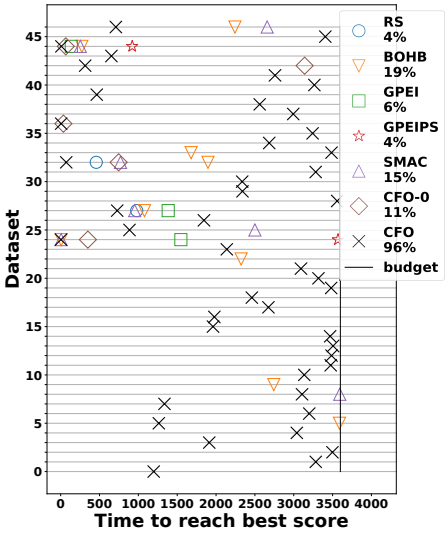
(a) Scores in tuning XGBoost



(b) Scores in tuning DNN



(c) Scores in tuning XGBoost, the higher the better. The legends display the fraction of datasets on which the scores are over 90 for each method



(d) Time used in reaching best score for XGBoost. The legends display the fraction of datasets on which the best score can be reached within 1h

Figure 2: Scores in (a)-(c) are normalized loss (100 for CFO, higher score for lower loss). The colored markers in (c) and (d) correspond to the cases where a baseline reaches score 99.95 within the time budget. The black markers in (c) do not have corresponding points in (d)

adequate. CFO−0 shares a similar spirit with CFO−0 because ZOGD can also be considered as a randomized direct search method. However, due to the unique designs in FLOW^2, CFO still maintains its leading performance. CFO demonstrates strong anytime performance, showing its effectiveness in controlling the evaluation cost incurred during the optimization process. It achieves up to three orders of magnitude speedup comparing to other methods for reaching any loss level.

Overall optimality of loss and cost. Figure 2a and 2b present boxplots of normalized scores (100 for CFO, the higher score for lower loss) obtained by all the baselines on all the datasets in tuning XGBoost and DNN within the required time budget. We can observe that CFO has dominating performance in terms of loss on the large collection of datasets under the same time budget with the others. To investigate the methods' optimality on each dataset, we show the normalized scores obtained on all the datasets for each method in tuning XGBoost within the required time budget in Figure 2c. We can see that CFO is able to find the best loss on almost all the datasets with a large margin comparing to the baselines. RS or each BO method has 19%-32% datasets (diversely distributed) with more than 10% gap in score compared to CFO. There are only two cases where CFO's score is lower than a baseline, i.e., BOHB, by 0.4%. Figure 2d shows the time for reaching the best loss of each dataset, which means reaching up to 0.05% below the highest score by all the compared methods within the required time budget in tuning XGBoost. These results show that (1) overall CFO has the highest ratio (96%) of reaching the best loss in contrast to the very low ratios in baselines (less than 19%); (2) on a very large fraction of the datasets, CFO can reach the best loss within a small amount of time while the others cannot reach the same loss within time budget. (3) in the cases where there are other methods reaching the best loss, CFO almost always uses the least amount of time.

Conclusion and Future Work

In this work, we take a novel path of addressing the HPO problem from the cost aspect, which is under-explored in existing literature but especially important. We consider our work one of the initial successful attempts to make HPO cost-frugal, having both dominating empirical performance and provable theoretical guarantee on the total cost. Our analysis of cost control is the first of its kind. It is a good starting point for better understandings of the cost behaviors of HPO solutions, including what conditions are needed for the HPO methods to be cost-frugal. As future work, it is worth studying the theoretical guarantees of our method under weaker conditions of the cost function. It is also worth studying how to effectively incorporate cost observations into the HPO algorithms to make it more frugal. In addition, one unique property of our method is that its iterative update does not require the exact difference of the compared configurations' performance and instead only needs the relative order of them. This unique property can potentially make CFO robust to noisy configuration evaluations and open up applications where relative orders of pairwise configurations performance comparison are easier or cheaper to obtain.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful and constructive comments. The authors would like to thank John Langford, Qiang Liu, Jian Peng, Hongning Wang and Markus Weimer for their advice.

Ethical Impact

Our work can help reduce the cost of doing hyperparameter optimization. It can be used to build more efficient automated machine learning (AutoML) solutions, which can save the effort and time of data scientists, and allow non-experts to make use of machine learning models and techniques. In a broader sense, we consider our work as an important attempt to make machine learning more economically and environmentally friendly. The current trend of massive consumption on computation resources in training and tuning machine learning models brings a tremendous burden to the environment. In fact, a recent study (Strubell, Ganesh, and McCallum 2020) quantified the approximate financial and environmental costs of training deep neural networks, which calls for methods that can reduce costs and improve equity in machine learning research and practice. Given this consideration, machine learning solutions should be designed to be cost-effective even if the computational budget is not a bottleneck for a specific task or a specific group of people. The cost-effective design of our method well aligns with this principle.

References

Bergstra, J.; and Bengio, Y. 2012. Random Search for Hyper-parameter Optimization. *J. Mach. Learn. Res.* 13: 281–305.

Bergstra, J. S.; Bardenet, R.; Bengio, Y.; and Kégl, B. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*.

Boyd, S.; Boyd, S. P.; and Vandenberghe, L. 2004. *Convex optimization*. Cambridge university press.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *KDD*.

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*.

Gijsbers, P.; LeDell, E.; Thomas, J.; Poirier, S.; Bischl, B.; and Vanschoren, J. 2019. An Open Source AutoML Benchmark. In *AutoML Workshop at ICML 2019*.

György, A.; and Kocsis, L. 2011. Efficient multi-start strategies for local search algorithms. *Journal of Artificial Intelligence Research* 41: 407–444.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In *Learning and Intelligent Optimization*.

Kandasamy, K.; Dasarathy, G.; Schneider, J.; and Póczos, B. 2017. Multi-fidelity bayesian optimisation with continuous approximations. In *Proceedings of the 34th International Conference on Machine Learning*.

Klein, A.; Falkner, S.; Bartels, S.; Hennig, P.; and Hutter, F. 2017. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In *Artificial Intelligence and Statistics*.

Kolda, T. G.; Lewis, R. M.; and Torczon, V. 2003. Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods. *SIAM Review* 45(3): 385–482.

Konnov, I. 2018. Conditional gradient method without line-search. *Russian Mathematics* 62(1): 82–85.

Li, L.; Jamieson, K.; DeSalvo, G.; Rostamizadeh, A.; and Talwalkar, A. 2017. Hyperband: A novel bandit-based approach to hyperparameter optimization. In *ICLR'17*.

Lu, Z.; Chen, L.; Chiang, C.-K.; and Sha, F. 2019. Hyper-parameter Tuning under a Budget Constraint. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*.

Maclaurin, D.; Duvenaud, D.; and Adams, R. 2015. Gradient-based Hyperparameter Optimization through Reversible Learning. In *Proceedings of the 32nd International Conference on Machine Learning*.

Martí, R.; Moreno-Vega, J. M.; and Duarte, A. 2010. Advanced multi-start methods. In *Handbook of metaheuristics*, 265–281. Springer.

Melis, G.; Dyer, C.; and Blunsom, P. 2018. On the State of the Art of Evaluation in Neural Language Models. In *International Conference on Learning Representations*.

Nesterov, Y.; and Spokoiny, V. 2017. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics* 17(2): 527–566.

Olson, R. S.; La Cava, W.; Orzechowski, P.; Urbanowicz, R. J.; and Moore, J. H. 2017. PMLB: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining* 10(1): 36.

Pedregosa, F. 2016. Hyperparameter Optimization with Approximate Gradient. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, ICML'16.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*.

Strubell, E.; Ganesh, A.; and McCallum, A. 2020. Energy and Policy Considerations for Modern Deep Learning Research. *Proceedings of the AAAI Conference on Artificial Intelligence* (09).

Wu, Q.; Wang, C.; and Huang, S. 2020. Cost Effective Optimization for Cost-related Hyperparameters. *arXiv preprint arXiv:2005.01571* .

Zabinsky, Z. B.; Bulger, D.; and Khompatraporn, C. 2010. Stopping and restarting strategy for stochastic sequential search in global optimization. *Journal of Global Optimization* 46(2): 273–286.