Non-stochastic Best Arm Identification and Hyperparameter Optimization

Kevin Jamieson
Electrical Engineering and Computer Science
University of California, Berkeley
kjamieson@eecs.berkeley.edu

Ameet Talwalkar
Computer Science
University of California, Los Angeles
ameet@cs.ucla.edu

Abstract

Motivated by the task of hyperparameter optimization, we introduce the *non-stochastic best-arm identification problem.* We identify an attractive algorithm for this setting that makes no assumptions on the convergence behavior of the arms' losses, has no free-parameters to adjust, provably outperforms the uniform allocation baseline in favorable conditions, and performs comparably (up to log factors) otherwise. Next, by leveraging the iterative nature of many learning algorithms, we cast hyperparameter optimization as an instance of non-stochastic best-arm identification. Our empirical results show that, by allocating more resources to promising hyperparameter settings, our approach achieves comparable test accuracies an order of magnitude faster than the uniform strategy. The robustness and simplicity of our approach makes it well-suited to ultimately replace the uniform strategy currently used in most machine learning software packages.

1 Introduction

As supervised learning methods are becoming more widely adopted, hyperparameter optimization has become increasingly important to simplify and speed up the development of data processing pipelines while simultaneously yielding more accurate models. In hyperparameter optimization for supervised learning, we are given labeled training data, a set of hyperparameters associated with our supervised learning task, e.g., kernel bandwidth, regularization constant, etc., and a search space over these hyperparameters. We aim to find a particular configuration of hyperparameters that optimizes some evaluation criterion, e.g., loss on a validation dataset.

Appearing in Proceedings of the 19th International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 41. Copyright 2016 by the authors.

Since many learning algorithms are iterative in nature, particularly when working at scale, we can evaluate the quality of intermediate results, i.e., partially trained models, resulting in a sequence of losses that converges to the final loss value at convergence. Figure 1 shows the sequence of validation losses for various hyperparameter settings for kernel SVM models trained via stochastic gradient descent. The figure shows high variability in model quality across hyperparameter settings, and it is natural to ask the question: *Can we identify and terminate poor-performing hyperparameter settings early in a principled online fashion to speed up hyperparameter optimization?*

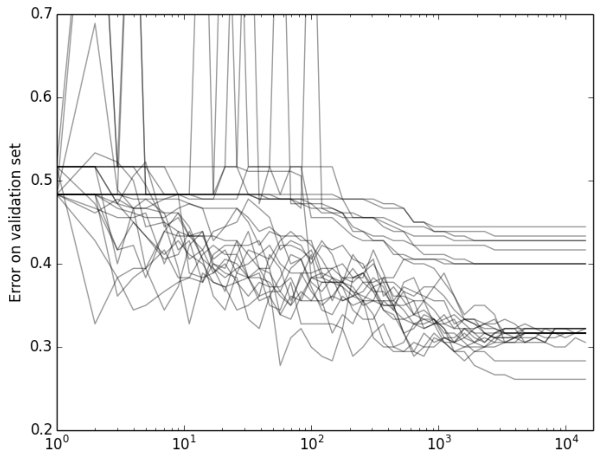


Figure 1: Validation error for various hyperparameter configurations for a classification task trained using stochastic gradient descent.

Although several hyperparameter optimization methods have been proposed recently, e.g., [2, 3, 4, 5, 6],

```
Best Arm Problem for Multi-armed Bandits
input: $n$ arms where $\ell_{i,k}$ denotes the loss observed on the $k$th pull of the $i$th arm
initialize: $T_i = 1$ for all $i \in [n]$
for $t = 1, 2, 3, \ldots$
    Algorithm chooses an index $I_t \in [n]$
    Loss $\ell_{I_t, T_{I_t}}$ is revealed, $T_{I_t} = T_{I_t} + 1$
    Algorithm outputs a recommendation $J_t \in [n]$
    Receive external stop signal, otherwise continue
```

Figure 2: A generalization of the best arm problem for multi-armed bandits [1] that applies to both the stochastic and non-stochastic settings.

the vast majority of them consider model training to be a black-box procedure, and only evaluate models after they are fully trained to convergence. A few recent works have made attempts to exploit intermediate results. However, these works either require explicit forms for the convergence rate behavior of the iterates which is difficult to accurately characterize for all but the simplest cases [7, 8], or focus on heuristics lacking theoretical underpinnings [9]. We build upon these previous works, and in particular study the multi-armed bandit formulation proposed in [7] and [9], where each arm corresponds to a fixed hyperparameter setting, pulling an arm corresponds to a fixed number of training iterations, and the loss corresponds to an intermediate loss on some hold-out set.

We aim to provide a robust, general-purpose, and widely applicable bandit-based solution to hyperparameter optimization. Remarkably, the existing multi-armed bandits literature fails to address this natural problem setting: a *non-stochastic best arm identification* problem. Indeed, existing work fails to adequately address the two main challenges in this setting: (i) We know that each arm's sequence of losses eventually converges, but we have no information about the rate of convergence, and the sequence of losses, like those in Figure 1, may exhibit a high degree of non-monotonicity and non-smoothness; (ii) The cost of obtaining the loss of an arm can be disproportionately more costly than pulling it, as in the case of hyperparameter optimization where computing the validation loss is often drastically more expensive than performing a single training iteration.

We note that the non-stochastic best arm setting is quite generally applicable, encompassing many problems including the stochastic best arm identification problem [1], less-well-behaved stochastic sources like max-bandits [10], exhaustive subset selection for feature extraction, and many optimization problems that "feel" like stochastic best arm problems but lack the i.i.d. assumptions necessary in that setting. For example, when minimizing a non-convex objective using gradient descent or some other iterative algorithm, it is common to perform random restarts. Instead of executing these restarts sequentially, it is natural to run several instances of the optimization procedure with different starting conditions in parallel, allocating resources dynamically across the instances based on their incremental performance as is done in [11]. However, while our work requires no knowledge of the rate at which each arm converges to its final limit, [11] assumes that the convergence rate is known up to an unknown constant common to all arms. For this reason, although [11] can be viewed as a relaxation of the settings of [7, 8], it is not a general-purpose solution like our work.

While there are several interesting applications of the non-stochastic best arm identification setting, we focus on the application of hyperparameter tuning. We first identify and analyze an algorithm particularly well-suited for this bandit setting due to its generality, robustness, and simplicity. We show theoretically that our adaptive allocation approach outperforms the uniform allocation baseline in favorable conditions, and performs comparably (up to log factors) otherwise. We then confirm our theory with empirical hyperparameter optimization studies that demonstrate an order of magnitude speedups relative to natural baselines on a number of supervised learning problems and datasets.

2 Non-stochastic best arm identification

While the multi-armed bandit objective of minimizing cumulative regret has been analyzed in both the stochastic and non-stochastic settings, to the best of our knowledge, the best arm objective has only been analyzed in the stochastic setting. In this section we present a formulation for the non-stochastic setting. Figure 2 presents a general form of the best arm problem for multi-armed bandits. Intuitively, at each time t the goal is to choose J_t such that the arm associated with J_t has the lowest loss in some sense. Note that while the algorithm gets to observe the value for an arbitrary arm I_t, the algorithm is only evaluated on its recommendation J_t, that it also chooses arbitrarily. We now consider the following two best arm identification settings:

Stochastic : For all $i \in [n]$, $k \geq 1$, let $\ell_{i,k}$ be an i.i.d. sample from a probability distribution on $[0, 1]$ such that $\mathbb{E}[\ell_{i,k}] = \mu_i$. The goal is to identify $\arg\min_i \mu_i$ while minimizing $\sum_{i=1}^n T_i$.

Non-stochastic (proposed in this work) : For all $i \in [n]$, $k \geq 1$, let $\ell_{i,k} \in \mathbb{R}$ be generated by an oblivious adversary, i.e., the loss sequences are independent of the algorithm's actions. Further, assume $\nu_i = \lim_{\tau \to \infty} \ell_{i,\tau}$ exists. The goal is to identify $\arg\min_i \nu_i$ while minimizing $\sum_{i=1}^n T_i$.

These two settings are related in that we can always turn the stochastic setting into the non-stochastic setting by defining $\ell_{i,T_i} = \frac{1}{T_i} \sum_{k=1}^{T_i} \ell'_{i,T_i}$ where ℓ'_{i,T_i} are the losses from the stochastic problem; by the law of large numbers $\lim_{\tau\to\infty} \ell_{i,\tau} = \mathbb{E}[\ell'_{i,1}]$. We could also take the minimum of the stochastic returns of an arm, as considered in [10], since $\ell_{i,T_i} = \min\{\ell'_{i,1}, \ell'_{i,2}, \dots, \ell'_{i,T_i}\}$ is a bounded, monotonically decreasing sequence, and consequently has a limit.

However, the generality of the non-stochastic setting introduces novel challenges. In the stochastic setting, if we set $\widehat{\mu}_{i,T_i} = \frac{1}{T_i} \sum_{k=1}^{T_i} \ell_{i,k}$ then $|\widehat{\mu}_{i,T_i} - \mu_i| \leq \sqrt{\log(4nT_i^2/\delta)/2T_i}$ for all $i \in [n]$ and $T_i > 0$ with probability at least $1-\delta$ by applying Hoeffding's inequality and a union bound. In contrast, the non-stochastic setting's assumption that $\lim_{\tau\to\infty} \ell_{i,\tau}$ exists implies that there exists a non-increasing function γ_i such that $|\ell_{i,t} - \lim_{\tau\to\infty} \ell_{i,\tau}| \leq \gamma_i(t)$ and that $\lim_{t\to\infty} \gamma_i(t) = 0$, but we know nothing about how *quickly* $\gamma_i(t)$ approaches 0 as a function of t. The lack of a convergence rate means that even the tightest $\gamma_i(t)$ could decay arbitrarily slowly and we would never know it.

This observation has two consequences. First, we can never *reject* the possibility that an arm is the "best" arm. Second, we can never *verify* that an arm is the "best" arm or even within $\epsilon > 0$ of the best arm. Without the possibility of these certificates, the problem is naturally one of a fixed budget: if a user has an hour to return an answer, she should run an algorithm for an hour and pick its recommendation. If she reaches the hour and has more time, she should run the algorithm longer. In this setting, the question is not *if* the algorithm will identify the best arm, but *how fast* it does so relative to a baseline method.

The natural baseline to compare to in our setting of interest is the default algorithm implemented in most software packages today, namely a uniform allocation strategy. Indeed, most packages including LibSVM [12], scikit-learn [13], and MLlib [14] train each hyperparameter setting to convergence. We are interested in comparing the performance of different hyperparameter settings throughout the training process and discarding poorly performing settings before convergence, thereby saving CPU-cycles and time. We propose a simple and intuitive algorithm that makes no assumptions on the convergence behavior of the losses, requires no inputs or free-parameters to adjust, provably outperforms the baseline method in favorable conditions and performs comparably (up to log factors) otherwise, and empirically identifies good hyperparameters an order of magnitude faster than the baseline method on a variety of problems. Given the generality of our approach, some problems may benefit from more special purpose solutions, however these approaches typically rely on strong assumptions that must be specified by an expert user, and even an expert may incorrectly specify them. In contrast, our goal is to design a general purpose algorithm that is robust and simple enough to ultimately replace the uniform allocation baseline in most popular software packages.

2.1 Related work

Despite dating back to the late 1950's, the best arm identification problem for the stochastic setting has experienced a surge of activity in the last decade. The work has two major branches: the fixed budget setting and the fixed confidence setting. In the fixed budget setting, the algorithm is given a set of arms and a budget B and is tasked with maximizing the probability of identifying the best arm by pulling arms without exceeding the total budget. While these algorithms were developed for and analyzed in the stochastic setting, they exhibit attributes that are amenable to the non-stochastic setting. In fact, the algorithm we propose to use in this paper is the Successive Halving algorithm of [15], though the non-stochastic setting requires its own novel analysis that we present in Section 3. Successive Rejects [16] is another fixed budget algorithm that we empirically evaluate.

In contrast, the fixed confidence setting takes an input $\delta \in (0,1)$ and guarantees to output the best arm with probability at least $1-\delta$ while attempting to minimize the number of total arm pulls. These algorithms, e.g., Successive Elimination [17], Exponential Gap Elimination [15], LUCB [18], and Lil'UCB [19], are ill-suited for the non-stochastic best arm identification problem because they rely on statistical bounds that are generally not applicable in the non-stochastic case.

Exploration algorithm	# observed losses
Uniform (baseline) (B)	n
Successive Halving (B)	$2n+1$
Successive Rejects (B)	$(n+1)n/2$
Successive Elimination (C)	$n\log_2(2B)$
LUCB (C), lil'UCB (C), EXP3 (R)	B

Table 1: Number of observed losses by the algorithm after B time steps and n number of arms. (B), (C), or (R) indicate fixed budget, fixed confidence, or cumulative regret, respectfully.

```
Successive Halving Algorithm
input: Budget $B$, $n$ arms where $\ell_{i,k}$ denotes the $k$th loss from the $i$th arm
Initialize: $S_0 = [n]$.
For $k = 0, 1, \ldots, \lceil \log_2(n) \rceil - 1$
    Pull each arm in $S_k$ for $r_k = \lfloor \frac{B}{|S_k| \lceil \log_2(n) \rceil} \rfloor$ additional times and set $R_k = \sum_{j=0}^{k} r_j$.
    Let $\sigma_k$ be a bijection on $S_k$ such that $\ell_{\sigma_k(1),R_k} \leq \ell_{\sigma_k(2),R_k} \leq \cdots \leq \ell_{\sigma_k(|S_k|),R_k}$
    $S_{k+1} = \{ i \in S_k : \ell_{\sigma_k(i),R_k} \leq \ell_{\sigma_k(\lfloor |S_k|/2 \rfloor),R_k} \}$.
output : Singleton element of $S_{\lceil \log_2(n) \rceil}$
```

Figure 3: The Successive Halving algorithm of [15].

In addition to the total number of arm pulls, we also consider the required number of *observed* losses. This is a natural cost to consider when ℓ_{i,T_i} for any i is the result of an expensive computation, e.g., computing validation error on a partially trained classifier. Assuming a known time horizon (or budget) B, Table 1 describes the total number of times various algorithms observe a loss as a function of B and the number of arms n. The EXP3 algorithm [20], a popular approach for minimizing cumulative regret in the non-stochastic setting, is also included. In practice $B \gg n$, and thus Successive Halving is a particularly attractive option, as along with the baseline, it is the only algorithm that observes losses proportional to the number of arms and independent of the budget. As shown in Section 5, empirical performance is quite dependent on the number of observed losses.

3 Proposed algorithm and analysis

The proposed Successive Halving algorithm of Figure 3 was originally introduced for stochastic best arm identification by [15]. However, our novel analysis shows that it is also effective in the non-stochastic setting. The idea behind the algorithm is simple: given an input budget, uniformly allocate the budget to a set of arms for a predefined amount of iterations, evaluate their performance, throw out the worst half, and repeat until just one arm remains.

The budget as an input is easily removed by the "doubling trick." This procedure sets $B \leftarrow n$, runs the algorithm to completion with B, then sets $B \leftarrow 2B$, and repeats ad infinitum. This method can reuse existing progress from round to round and effectively makes the algorithm parameter-free. Notably, if a budget of B' is necessary to find the best arm, then by using the doubling trick we can find the best arm with a budget of $2B'$ in the worst case without knowing B' in the first place.

3.1 Analysis of Successive Halving

For $i = 1, \ldots, n$ define $\nu_i = \lim_{\tau \to \infty} \ell_{i,\tau}$ which exists by assumption. Without loss of generality, assume $\nu_1 < \nu_2 \leq \cdots \leq \nu_n$. We next introduce functions that bound the approximation error of $\ell_{i,t}$ with respect to ν_i as a function of t. For each $i \in [n]$ let $\gamma_i(t)$ be the point-wise smallest, non-increasing function of t with $|\ell_{i,t} - \nu_i| \leq \gamma_i(t) \quad \forall t$. In addition, define $\gamma_i^{-1}(\alpha) = \min\{t \in \mathbb{N} : \gamma_i(t) \leq \alpha\}$ for all $i \in [n]$. With this definition, if $t_i > \gamma_i^{-1}(\frac{\nu_i - \nu_1}{2})$ and $t_1 > \gamma_1^{-1}(\frac{\nu_i - \nu_1}{2})$ then

$$\begin{aligned}\ell_{i,t_i} - \ell_{1,t_1} &= (\ell_{i,t_i} - \nu_i) + (\nu_1 - \ell_{1,t_1}) + 2\left(\tfrac{\nu_i - \nu_1}{2}\right) \\ &\geq -\gamma_i(t_i) - \gamma_1(t_1) + 2\left(\tfrac{\nu_i - \nu_1}{2}\right) > 0\end{aligned}$$

so that $\ell_{i,t_i} > \ell_{1,t_1}$. That is, comparing the intermediate values at t_i and t_1 suffices to determine the ordering of the final values ν_i and ν_1. Intuitively, this condition holds because the envelopes at the given times, namely $\gamma_i(t_i)$ and $\gamma_1(t_1)$, are small relative to the gap between ν_i and ν_1. This line of reasoning is at the heart of the proof of our main result, and the theorem is stated in terms of these quantities. All proofs can be found in the appendix.

Theorem 1 *Let* $\nu_i = \lim_{\tau \to \infty} \ell_{i,\tau}$, $\bar{\gamma}(t) = \max_{i=1,\ldots,n} \gamma_i(t)$, *and*

$$\begin{aligned}z_{SH} &= 2\lceil \log_2(n) \rceil \max_{i=2,\ldots,n} i \left(1 + \bar{\gamma}^{-1}\left(\tfrac{\nu_i - \nu_1}{2}\right)\right) \\ &\leq 2\lceil \log_2(n) \rceil \Big(n + \sum_{i=2,\ldots,n} \bar{\gamma}^{-1}\left(\tfrac{\nu_i - \nu_1}{2}\right)\Big).\end{aligned}$$

If the Successive Halving algorithm is run with any budget $B > z_{SH}$ *then the best arm is guaranteed to be returned. Moreover, if the Successive Halving algorithm is bootstrapped by the "doubling trick" that takes no arguments as input, then this procedure returns the best arm once the total number of iterations taken exceeds just* $2z_{SH}$.

Remark 1 *Without additional assumptions on the* γ_i *functions, it is impossible for* any *algorithm to know with certainty that an arm is the best arm, and thus when it can stop and output an arm with any confidence. Consequently, among the set of algorithms that are guaranteed to identify the best arm* eventually, *our burden is to characterize situations in which the Successive Halving algorithm identifies the best-arm significantly faster than natural baselines, and show that it is never much slower. In other words, for any amount of time allotted to a search process, we must show that Successive Halving is the preferred algorithm in hindsight with respect to natural baselines. Note that any heuristic optimization convergence criteria (e.g. a lack of progress in the last several iterations) can be applied here just as is often done in practice.*

The representation of z_{SH} on the right-hand-side of the inequality is intuitive: if $\bar{\gamma}(t) = \gamma_i(t) \quad \forall i$ and an oracle gave us an explicit form for $\bar{\gamma}(t)$, then to merely verify that the ith arm's final value is higher than the best arm's value, one must pull each of the two arms at least a number of times equal to the ith term in the sum (this becomes clear by inspecting the proof of Theorem 3). Repeating this argument for all $i = 2, \ldots, n$ explains the sum over all $n-1$ arms.

Example 1 *Consider a feature-selection problem involving a dataset $\{(x_i, y_i)\}_{i=1}^n$ where each $x_i \in \mathbb{R}^D$, with the goal of identifying the best subset of d features that linearly predicts y_i in terms of the least-squares metric. Defining each d-subset is an arm we have $n = \binom{D}{d}$ arms. The least squares problem can be efficiently solved with stochastic gradient descent. Using known bounds for the rates of convergence [21] one can show that $\gamma_a(t) \leq \frac{\sigma_a \log(nt/\delta)}{t}$ for all $a = 1, \ldots, n$ arms and all $t \geq 1$ with probability at least $1 - \delta$ where σ_a is a constant that depends on the condition number of the quadratic defined by the d-subset. Then in Theorem 1, $\bar{\gamma}(t) = \frac{\sigma_{\max} \log(nt/\delta)}{t}$ with $\sigma_{\max} = \max_{a=1,\ldots,n} \sigma_a$ so after inverting $\bar{\gamma}$ we find that $z_{SH} = 4\lceil \log_2(n) \rceil \max_{a=2,\ldots,n} a \frac{4\sigma_{\max} \log\left(\frac{2n\sigma_{\max}}{\delta(\nu_a - \nu_1)}\right)}{\nu_a - \nu_1}$ is a sufficient budget to identify the best arm.*

In this example we computed upper bounds on γ_i in terms of problem dependent parameters and plugged them into our theorem to obtain a sample complexity. However, we stress that constructing tight bounds for the γ_i functions is very difficult outside of simple problems, and even then we have unspecified constants that require expert knowledge to approximate. Although such special purpose approaches could yield more efficient algorithms, they often require the careful tuning of additional hyperparameters and can be challenging for non-experts to deploy.

In contrast, our algorithm does not explicitly rely on these γ_i functions, and is in some sense *adaptive* to them: the faster the arms' losses converge, the faster the best arm is discovered, without ever changing the algorithm. This behavior is in contrast to the hyperparameter tuning work of [8] and [7], in which the algorithms explicitly take upper bounds on these γ_i functions as input, meaning their accuracy and performance is only as good as the tightness of these difficult to calculate bounds.

3.2 Comparison to a uniform allocation strategy

We next derive a result for the uniform allocation strategy that pulls arms in a round-robin fashion.

Theorem 2 *(Uniform strategy – sufficiency) Let $\nu_i = \lim_{\tau \to \infty} \ell_{i,\tau}$, $\bar{\gamma}(t) = \max_{i=1,\ldots,n} \gamma_i(t)$ and*

$$z_U = \max_{i=2,\ldots,n} n\bar{\gamma}^{-1}\left(\tfrac{\nu_i - \nu_1}{2}\right).$$

The uniform strategy takes no input arguments and returns the best arm at timestep t for all $t > z_U$.

Theorem 2 is a sufficiency statement so it is unclear how it actually compares to the Successive Halving result of Theorem 1. The next theorem says that the above result is tight in a worst-case sense, exposing the real gap between Successive Halving and uniform allocation.

Theorem 3 *(Uniform strategy – necessity) For any timestep t and final values $\nu_1 < \nu_2 \leq \cdots \leq \nu_n$ there exists a sequence of losses $\{\ell_{i,t}\}_{t=1}^\infty$, $i = 1, 2, \ldots, n$ such that if*

$$t < \max_{i=2,\ldots,n} n\bar{\gamma}^{-1}\left(\tfrac{\nu_i - \nu_1}{2}\right)$$

then the uniform strategy does not return the best arm at timestep t.

Remark 2 *If we consider the second, looser representation of z_{SH} on the right-hand-side of the inequality in Theorem 1 and multiply this quantity by $\frac{n-1}{n-1}$ we see that the sufficient number of pulls for the Successive Halving algorithm with doubling essentially behaves like $(n-1)\log_2(n)$ times the* average $\frac{1}{n-1}\sum_{i=2,\ldots,n} \bar{\gamma}^{-1}\left(\frac{\nu_i - \nu_1}{2}\right)$ *whereas the necessary result of the uniform strategy of Theorem 3 behaves like n times the* maximum $\max_{i=2,\ldots,n} \bar{\gamma}^{-1}\left(\frac{\nu_i - \nu_1}{2}\right)$. *The next example shows that the difference between this average and max can be significant.*

Example 2 *Recall Example 1 and now assume that $\sigma_a = \sigma_{\max}$ for all $a = 1, \ldots, n$. Then Theorem 3 says that the uniform strategy budget must be at least $n \frac{4\sigma_{\max} \log\left(\frac{2n\sigma_{\max}}{\delta(\nu_2 - \nu_1)}\right)}{\nu_2 - \nu_1}$ to identify the best arm. To see how this result compares with that of Successive Halving with doubling, let us parameterize the ν_a limiting values such that $\nu_a = a/n$ for $a = 1, \ldots, n$. Then a sufficient budget for the Successive Halving algorithm with doubling to identify the best arm is just $16n\lceil \log_2(n) \rceil \sigma_{\max} \log\left(\frac{n^2\sigma_{\max}}{\delta}\right)$ while the uniform strategy would require a budget of at least $2n^2\sigma_{\max} \log\left(\frac{n^2\sigma_{\max}}{\delta}\right)$. This is a difference of essentially $8n\log_2(n)$ versus n^2.*

3.3 Anytime fallback guarantee

It was just shown that Successive Halving can potentially identify the best arm well before the uniform procedure, i.e., $z_{SH} \ll z_U$. However, z_{SH} is in terms of $\gamma_i(t)$ and is usually not available. It is natural to ask what happens if we stop Successive Halving before these results apply, i.e., when $t < z_{SH}$. Our next result, an anytime performance guarantee, answers this question by showing that in such cases Successive Halving is comparable to the baseline method modulo log factors.

Theorem 4 *Let $\widehat{i}_{SH}$ be the output of Successive Halving with doubling at timestep t. Then*

$$\nu_{\widehat{i}_{SH}} - \nu_1 \leq \lceil \log_2(n) \rceil 2\bar{\gamma}\left(\lfloor \tfrac{t/2}{n\lceil \log_2(n)\rceil} \rfloor\right).$$

Moreover, $\widehat{i}_U$, the output of the uniform strategy at time t, satisfies

$$\nu_{\widehat{i}_U} - \nu_1 \leq \ell_{\widehat{i},B/n} - \ell_{1,B/n} + 2\bar{\gamma}(\lfloor t/n \rfloor) \leq 2\bar{\gamma}(\lfloor t/n \rfloor).$$

Applying Theorem 4 to the specific problem studied in Examples 1 and 2 shows that both Successive Halving and uniform allocation satisfy $\widehat{\nu}_i - \nu_1 \leq \widetilde{O}\left(n/B\right)$ in this particular setting, where $\widehat{i}$ is the output of either algorithm and $\widetilde{O}$ suppresses poly log factors. However, we stress that this result is merely a fall-back guarantee and it does not rule out the possibility of Successive Halving far outperforming uniform allocation in practice, as is suggested by Remark 2 and is observed in our experimental results.

4 Hyperparameter optimization and best arm identification

In supervised learning we are given a dataset of pairs $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ for $i = 1, \dots, n$ sampled i.i.d. from some unknown distribution $P_{X,Y}$, and we aim to find a map (or model) $f : \mathcal{X} \to \mathcal{Y}$ that minimizes $\mathbb{E}_{(X,Y)\sim P_{X,Y}}\left[\texttt{loss}(f(X), Y)\right]$ for some known loss function $\texttt{loss} : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$. We consider mappings that are the output of some fixed, possibly randomized, algorithm $\mathcal{A}$ that takes a dataset and algorithm-specific parameters $\theta \in \Theta$ as input and outputs $f_\theta : \mathcal{X} \to \mathcal{Y}$. For a fixed dataset $\{(x_i, y_i)\}_{i=1}^n$ the parameters $\theta \in \Theta$ index the different models f_θ, and will henceforth be referred to as *hyperparameters*. We adopt the train-validate-test framework for choosing hyperparameters [22], whereby we partition the full dataset into `TRAIN`, `VAL` , and `TEST` sets; use `TRAIN` to train a model $f_\theta = \mathcal{A}\left(\{(x_i, y_i)\}_{i\in\texttt{TRAIN}}, \theta\right)$ for each $\theta \in \Theta$; choose the hyperparameters that minimize the validation error; and report the test error. We note that the selection of $\widehat{\theta}$ is the minimization of a function that in general is not necessarily even continuous, much less convex.

Example 3 *Consider a linear support vector machine (SVM) classification example where $\mathcal{X} \times \mathcal{Y} = \mathbb{R}^d \times \{-1, 1\}$, $\Theta \subset \mathbb{R}_+$, $f_\theta = \mathcal{A}\left(\{(x_i, y_i)\}_{i\in\texttt{TRAIN}}, \theta\right)$ where $f_\theta(x) = sign\{\langle w_\theta, x\rangle\}$ with $w_\theta = \arg\min_w \frac{1}{|\texttt{TRAIN}|}\sum_{i\in\texttt{TRAIN}} \max(0, 1 - y_i\langle w, x_i\rangle) + \theta||w||_2^2$, and finally $\widehat{\theta} = \arg\min_{\theta\in\Theta} \frac{1}{|\texttt{VAL}|}\sum_{i\in\texttt{VAL}} \mathbf{1}\{y \neq f_\theta(x)\}$. While f_θ for each $\theta \in \Theta$ can be efficiently computed using an iterative algorithm due to convexity [23], the selection of $\widehat{\theta}$ is the minimization of a function that is not even continuous, much less convex.*

Now assume that the algorithm $\mathcal{A}$ is iterative so that for a given $\{(x_i, y_i)\}_{i\in\texttt{TRAIN}}$ and θ, the algorithm outputs a function $f_{\theta,t}$ every iteration $t > 1$. Define $\ell_{\theta,t}$ as the validation error of $f_{\theta,t}$. We assume that $\lim_{t\to\infty} \ell_{\theta,t}$ exists[1] and is equal to $\frac{1}{|\texttt{VAL}|}\sum_{i\in\texttt{VAL}} \texttt{loss}(f_\theta(x_i), y_i)$. Under these conditions, we can cast hyperparameter optimization as an instance of non-stochastic best arm identification. We generate the arms (hyperparameter settings) uniformly at random (possibly on a log scale) from within the region of valid hyperparameters (i.e. all hyperparameters within some minimum and maximum range) and sample enough arms to ensure a sufficient cover of the space [6]. Alternatively, one could input a fixed set of parameters of interest. We note that random or grid search remain the default choices for many open source machine learning packages such as LibSVM [12], scikit-learn [13] and MLlib [14]. As described in Figure 2, the bandit algorithm will choose I_t, and we will use the convention that $J_t = \arg\min_\theta \ell_{\theta,T_\theta}$. The final quality of the arm selected by J_t will be evaluated via test error as described above.

4.1 Related work

We aim to leverage the iterative nature of standard learning algorithms to speed up hyperparameter optimization in a robust and principled fashion. It is clear that no algorithm can provably identify a hyperparameter with a value within ϵ of the optimal without known, explicit functions γ_i, which means no algorithm can reject a hyperparameter setting with absolute confidence without making potentially strong assumptions. In [8], γ_i functions are defined in an ad-hoc, algorithm-specific, and data-specific fashion which leads to strong ϵ-good claims. A related line of work defines γ_i-like functions for optimizing the computational efficiency of structural risk minimization, yielding bounds [7]. We stress that these results are only as good as the tightness and correctness of the γ_i bounds. If the γ_i functions are chosen to decrease too rapidly, a procedure might throw out good arms too early, and if chosen to decrease too slowly, a procedure

[1]We note that $f_\theta = \lim_{t\to\infty} f_{\theta,t}$ pointwise on $\mathcal{X}$ is not enough to conclude that $\lim_{t\to\infty} \ell_{\theta,t}$ exists but we ignore this technicality in our experiments.

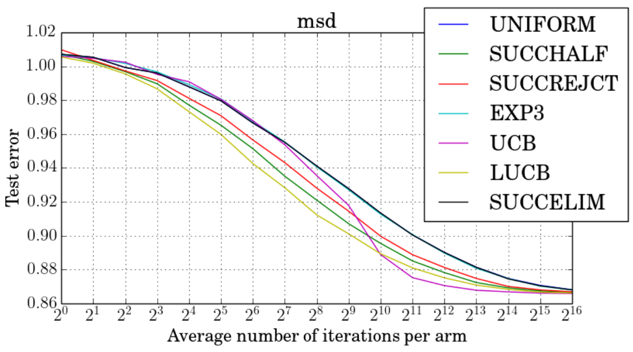
will be overly conservative. Moreover, properly tuning these special-purpose approaches can be an onerous task for non-experts. We view our work as an empirical, data-driven driven approach to the pursuits of [7]. Also, [9] empirically studies an early stopping heuristic similar in spirit to Successive Halving.

We also note that we fix the hyperparameter settings (or arms) under consideration and adaptively allocate our budget to each arm. In contrast, Bayesian optimization advocates choosing hyperparameter settings adaptively, but with the exception of [8], allocates a fixed budget to each selected hyperparameter setting [2, 3, 4, 5, 6]. These methods, though heuristic in nature as they attempt to simultaneously fit and optimize a non-convex and potentially high-dimensional function, yield promising empirical results. We view our approach as complementary and, as formulated, incomparable to Bayesian methods since our core interest is in exploiting the iterative nature of learning algorithms while these Bayesian methods treat these algorithms as black boxes. We discuss extensions of this work that would allow for a direct comparison to Bayesian methods in Section 6.

5 Experiment results

In this section we compare the proposed algorithm to a number of other algorithms, including the baseline uniform allocation strategy, on a number of hyperparameter optimization problems using the experimental setup outlined in Section 4. Each experiment was implemented in Python on an Amazon EC2 c3.8xlarge instance with 32 cores and 60 GB of memory. All datasets were partitioned into a 72-18-10 `TRAIN-VAL-TEST` split, and all plots report test error. To evaluate search algorithms, we fix a total budget of iterations and allow the search algorithms to allocate this budget amongst the different arms. The curves are produced by implementing the doubling trick by doubling the measurement budget each time. For the purpose of interpretability we do not warm start upon doubling. All datasets, aside from the collaborative filtering experiments, are normalized so that each dimension has mean 0 and variance 1.

Ridge Regression We first consider using stochastic gradient descent on the ridge regression objective function with step size $.01/\sqrt{2+T_\lambda}$. The ℓ_2 penalty hyperparameter $\lambda \in [10^{-6}, 10^0]$ was chosen uniformly at random on a log scale per trial, with 10 values (i.e., arms) selected per trial. We use the Million Song Dataset year prediction task [24] where we have downsampled the dataset by a factor of 10. The experiment was repeated for 32 trials using mean-squared error as the loss function. In the top panel of Figure 4 we note that



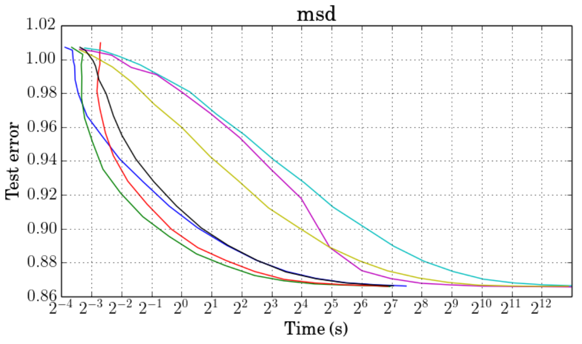


Figure 4: Ridge Regression. Test error with respect to both the number of iterations (top) and wall-clock time (bottom). Note that in the top plot, uniform, EXP3, and Successive Elimination coincide.

lil'UCB achieves a small test error two to four times faster in terms of iterations than most other methods. However, in the bottom panel the same data is plotted but with respect to wall-clock time rather than iterations and we observe that Successive Halving and Successive Rejects are the top performers. This results follows from Table 1, as lil'UCB must evaluate the validation loss on every iteration requiring much greater compute time. This pattern is observed in all experiments so in the sequel we only consider uniform allocation, Successive Halving, and Successive Rejects.

Collaborative filtering We next consider a matrix completion problem using the Movielens 100k dataset trained with stochastic gradient descent on the bi-convex objective with step sizes as described in [25]. We initialize the user and item variables with entries drawn from a normal distribution with variance σ^2/d, hence each arm has hyperparameters d (rank), λ (Frobenius norm regularization), and σ (initial conditions which may yield different locally optimal solutions). We chose $d \in [2, 50]$ and $\sigma \in [.01, 3]$ uniformly at random from a linear scale, and $\lambda \in [10^{-6}, 10^0]$ uniformly at random on a log scale. Each hyperparameter is given 4 samples resulting in $4^3 = 64$ total arms. We performed 32 trials and used mean-squared error as the loss function. Figure 5 shows that uniform allocation takes two to eight times longer in both wall-clock time and iterations to achieve a particular error rate than Successive Halving or Successive Rejects.

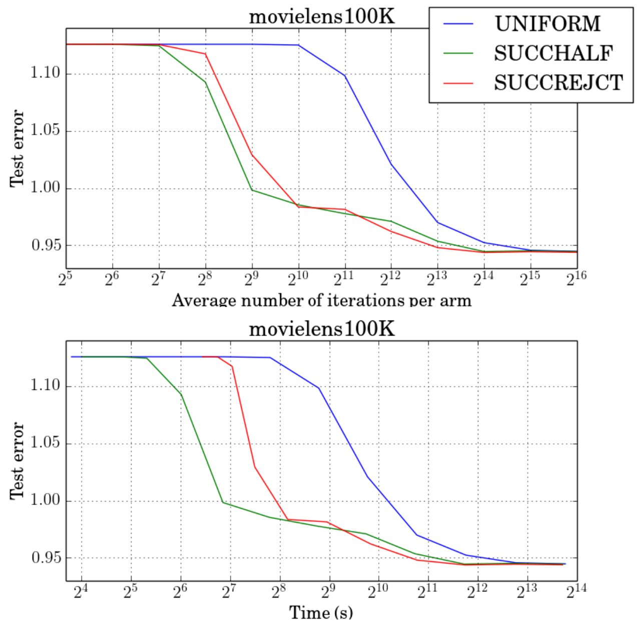


Figure 5: Matrix Completion (bi-convex formulation). Test error with respect to both the number of iterations (top) and wall-clock time (bottom). Successive Halving is roughly 4× faster than uniform in terms of both iterations and wall-clock time.

Kernel SVM We now consider learning a RBF-kernel SVM using Pegasos [23], with ℓ_2 penalty hyperparameter $\lambda \in [10^{-6}, 10^{0}]$ and kernel width $\gamma \in [10^{0}, 10^{3}]$ both chosen uniformly at random on a log scale per trial. Each hyperparameter was allocated 10 samples resulting in $10^2 = 100$ total arms. The experiment was repeated for 64 trials using 0/1 loss. Kernel evaluations were computed online (i.e. not precomputed and stored). We observe in Figure 6 that Successive Halving obtains the same low error more than an order of magnitude faster than both uniform and Successive Rejects with respect to wall-clock time.

6 Future directions

We see many future interesting theoretical, algorithmic, and empirical extensions of our work. First, we conjecture that a matching lower bound to Theorem 1 can be derived by considering a stochastic adversary and appealing to the methods of [26]. Since Theorem 1 is in terms of $\max_i \gamma_i(t)$, such a lower bound would be in stark contrast to the stochastic setting, where arms with smaller variances have smaller envelopes and existing algorithms exploit this fact [27]. Second, we plan to incorporate pairwise switching costs into our framework to model the degree to which resources are shared

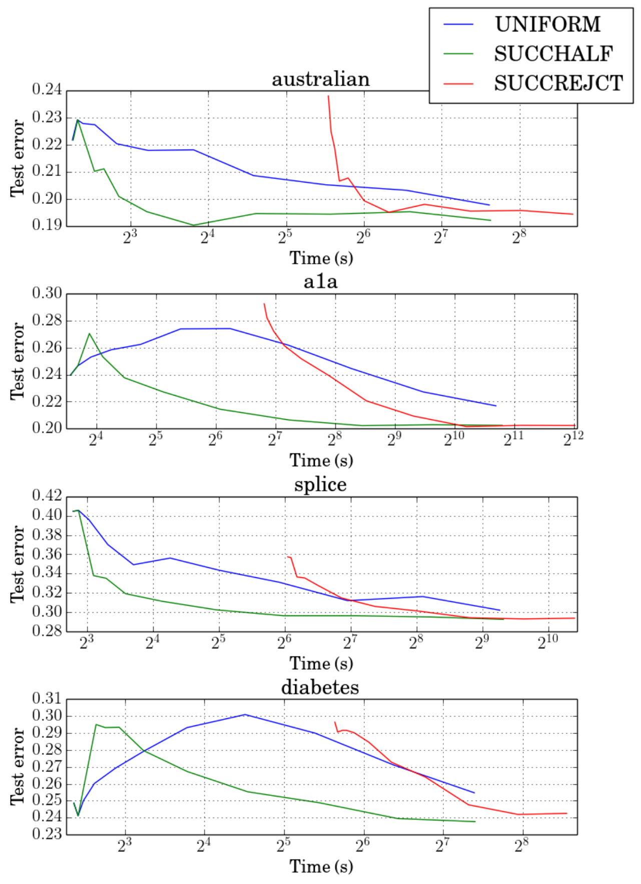


Figure 6: Kernel SVM. While Successive Halving and Successive Rejects perform similarly with respect to iteration count (not shown), they are separated by an order of magnitude in terms of wall-clock time.

across models (resulting in lower switching costs) on a single core machine. Exploring different ways of parallelizing the Successive Halving algorithm is also of considerable interest. Finally, we aim to extend our framework to work with black box solvers that return fully trained models. In such a setting, arms would still correspond to hyperparameter settings, while the number of pulls would correspond to the number of training examples provided to the solver. Enabling the use of subsampling to speed up hyperparameter tuning would be particularly attractive when working with solvers that have superlinear runtime complexity.

7 Acknowledgments

KJ is generously supported by ONR awards N00014-15-1-2620 and N00014-13-1-0129. AT is supported in part by a Google Faculty Award and an AWS in Education Research Grant award.

References

[1] Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Algorithmic Learning Theory*, pages 23–37. Springer, 2009.

[2] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

[3] Jasper Snoek, Kevin Swersky, Richard Zemel, and Ryan Adams. Input warping for bayesian optimization of non-stationary functions. In *ICML*, 2014.

[4] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. 2011.

[5] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. *NIPS*, 2011.

[6] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *JMLR*, 2012.

[7] Alekh Agarwal, Peter Bartlett, and John Duchi. Oracle inequalities for computationally adaptive model selection. *COLT*, 2011.

[8] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Freeze-thaw bayesian optimization. *arXiv:1406.3896*, 2014.

[9] Evan R Sparks, Ameet Talwalkar, Michael J. Franklin, Michael I. Jordan, and Tim Kraska. TuPAQ: An efficient planner for large-scale predictive analytic queries. *In Symposium on Cloud Computing*, 2015.

[10] Vincent A Cicirello and Stephen F Smith. The max k-armed bandit: A new model of exploration applied to search heuristic selection. In *National Conference on Artificial Intelligence*, volume 20, 2005.

[11] András György and Levente Kocsis. Efficient multi-start strategies for local search algorithms. *JAIR*, 41, 2011.

[12] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2, 2011.

[13] F. Pedregosa et al. Scikit-learn: Machine learning in Python. *JMLR*, 12, 2011.

[14] B. Yavuz E. Sparks S. Venkataraman D. Liu J. Freeman D. Tsai M. Amde S. Owen D. Xin R. Xin M. J. Franklin R. Zadeh M. Zaharia A. Talwalkar X. Meng, J. Bradley. MLlib: Machine learning in apache spark. *JMLR-MLOSS*, 2015.

[15] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *ICML*, 2013.

[16] Jean-Yves Audibert and Sébastien Bubeck. Best arm identification in multi-armed bandits. In *COLT-23th Conference on Learning Theory-2010*, pages 13–p, 2010.

[17] Eyal Even-Dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for the multi-armed bandit and reinforcement learning problems. *JMLR*, 7, 2006.

[18] Shivaram Kalyanakrishnan, Ambuj Tewari, Peter Auer, and Peter Stone. Pac subset selection in stochastic multi-armed bandits. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 655–662, 2012.

[19] Kevin Jamieson, Matthew Malloy, Robert Nowak, and Sébastien Bubeck. lil'ucb: An optimal exploration algorithm for multi-armed bandits. In *COLT*, 2014.

[20] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32(1):48–77, 2002.

[21] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

[22] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[23] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[24] M. Lichman. UCI machine learning repository, 2013.

[25] Benjamin Recht and Christopher Ré. Parallel stochastic gradient algorithms for large-scale matrix completion. *Mathematical Programming Computation*, 5(2):201–226, 2013.

[26] Emilie Kaufmann, Olivier Cappé, and Aurélien Garivier. On the complexity of best arm identification in multi-armed bandit models. *JMLR*, 2015.

[27] Aurélien Garivier and Olivier Cappé. The kl-ucb algorithm for bounded stochastic bandits and beyond. 2011.