# Simple (dynamic) bandit algorithms for hyper-parameter optimization

**Xuedong Shang** xuedong.shang@inria.fr
SequeL team, INRIA Lille - Nord Europe
**Emilie Kaufmann** emilie.kaufmann@univ-lille.fr
CNRS & ULille, UMR 9189 (CRIStAL), SequeL team, INRIA Lille - Nord Europe
**Michal Valko** valkom@google.com
DeepMind Paris

## Abstract

Hyper-parameter tuning is a major part of modern machine learning systems. The tuning itself can be seen as a sequential resource allocation problem. As such, methods for multi-armed bandits have been already applied. In this paper, we view hyper-parameter optimization as an instance of best-arm identification in infinitely many-armed bandits. We propose D-TTTS, a new adaptive algorithm inspired by Thompson sampling, which *dynamically* balances between refining the estimate of the quality of hyper-parameter configurations previously explored and adding new hyper-parameter configurations to the pool of candidates. We further provide insights on when and why D-TTTS shall work. The algorithm is easy to implement and shows competitive performance compared to state-of-the-art algorithms for hyper-parameter tuning.

## 1. Introduction

Training a machine learning algorithm often requires to specify several parameters. For instance, for neural networks, it is the architecture of the network and also the parameters of the gradient algorithm used or the choice of regularization. These *hyper-parameters* are difficult to learn through the standard training process and are often manually specified.

When it is not feasible to design algorithms with a few hyper-parameters, we opt for *hyper-parameter optimization* (HPO). HPO can be viewed as a *black-box optimization* problem where the evaluation of the objective function is expensive as it is the accuracy of a learning algorithm for a given configuration of hyper-parameters. This vastly limits the number of evaluations that can be carried out, which calls for a design of efficient high-level algorithms that automate the tuning procedure.

Several naive but daily used HPO methods are grid search and random search. More sophisticated methods address HPO as a *sequential resource allocation problem*, by adaptively choosing the next hyper-parameter(s) to explore, based on the result obtained previously. For example, *evolutionary optimization* follows a process inspired by the biological concept of *evolution*, which repeatedly replaces the worst-performing hyper-parameter configurations from a randomly initialized population of solutions; see Loshchilov and Hutter (2016) for an example of using CMA-ES for hyper-parameter tuning. A major drawback of evolutionary optimization is its lack of theoretical understanding.

Bayesian optimization (BO) is another approach that leverages the sequential nature of the setting. BO depends on a prior belief for the target function, typically a Gaussian process. This prior distribution can be updated to a posterior given a sequence of observations. Several algorithms exploiting this posterior distribution to decide where to sample next have been given (see Shahriari et al., 2016, for a survey). Snoek et al. (2012) and Klein et al. (2017) provide Python packages called `Spearmint` and `RoBO` to perform hyper-parameter tuning with BO methods. Similar packages are available for `PyTorch` (`BoTorch`[1]) and `TensorFlow` (`GPflowOpt` by Knudde et al. 2017). Among BO algorithms, `TPE` (Bergstra et al., 2011) and `SMAC` (Hutter et al., 2011) were specifically proposed for HPO. A shortcoming of BO is that most algorithms select where to sample next based on optimizing some *acquisition function* computed from the posterior, e.g., the expected improvement (Jones et al., 1998). This auxiliary task cannot be solved analytically but needs to be performed itself by optimization procedures as `L-BFGS` that make the process slow.

Bandits (see Lattimore and Szepesvari, 2018 for a recent book) are a simple model for sequential resource allocation, and some bandit tools have already been explored for global optimization and HPO: First, in the field of Bayesian optimization, the `GP-UCB` algorithm (Srinivas et al., 2010) is a Gaussian process extension of the classical `UCB` bandit algorithm (Auer et al., 2002). Later, Hoffman et al. (2014) proposed to use *best-arm identification* (BAI) tools—still with a Bayesian flavor—for automated machine learning, where the goal is to smartly try hyper-parameters from a pre-specified *finite* grid.

However, in most cases, the number of hyper-parameter configurations to explore is infinite. In this paper, we investigate the use of bandit tools suited for an *infinite* number of arms. There are two lines of work for tackling a very large or infinite number of configurations (arms). The first combines standard bandit tools with a hierarchical partitioning of the arm space and aims at exploiting the (possibly unknown) smoothness of the black-box function to optimize (Bubeck et al., 2010; Grill et al., 2015; Shang et al., 2019; Bartlett et al., 2019). To the best of our knowledge, these methods have never been investigated for HPO. The second line of work does not assume any smoothness: At each round, the learner may ask for a new arm from a *reservoir distribution* $\nu_0$ (pick randomly a new hyper-parameter configuration) and add it to the current arm pool $\mathcal{A}$, or re-sample one of the previous arms (evaluate configuration already included in $\mathcal{A}$), in order to find an arm with a good mean reward (i.e., a hyper-parameter configuration with a good validation accuracy). The *stochastic infinitely many-armed bandits* (SIAB) is studied by Berry et al. (1997); Wang et al. (2008) for the rewards maximization problem while Carpentier and Valko (2015); Aziz et al. (2018a) study the simple regret problem, which is related to BAI. While most proposed algorithms consist of querying an *adequate* number of arms from the reservoir before running a standard BAI algorithm, Li et al. (2017) propose a more robust approach called `Hyperband` that uses several such phases.

In this paper, we go even further and propose the first *dynamic* algorithm for BAI in SIAB, that at each round, may either query a new arm from the reservoir or re-sample arms previously queried. Our algorithm leverages a Bayesian model and builds on the top-two Thompson sampling (`TTTS`) algorithm by Russo (2016). An extensive numerical study is presented to show the competitiveness of our algorithm with respect to state-of-the-art HPO methods.

1. https://botorch.org/

## 2. Hyper-parameter optimization framework

In this paper, we view HPO as a particular *global optimization* setting, for which the target function $f$ is a mapping from a hyper-parameter configuration to some measure of failure for the machine learning algorithm trained with these hyper-parameters. Formally, we aim at solving an optimization problem of the form $f^\star = \min\{f(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \Omega\}$, where $\boldsymbol{\lambda}$ denotes a configuration of hyper-parameters chosen from a configuration space $\Omega$. A hyper-parameter optimizer is a sequential procedure, that at each round $t$, selects a configuration $\boldsymbol{\lambda}_t$ to evaluate using some sampling rule, after which a (costly and *noisy*) evaluation of $f(\boldsymbol{\lambda}_t)$ is observed. Besides, a hyper-parameter configuration $\widehat{\boldsymbol{\lambda}}^\star$ is recommended as a guess for a close-to-optimal configuration at the end. The hope is that $f(\widehat{\boldsymbol{\lambda}}^\star)$ is not far from $f^\star$.

We restrict our presentation to hyper-parameter tuning for supervised learning algorithms. Given a training dataset $\mathcal{D}_{\texttt{train}}$ containing $n$ labeled examples in $\mathcal{X} \times \mathcal{Y}$ and a choice of hyper-parameter configuration $\boldsymbol{\lambda}$, a supervised learning algorithm (neural network, SVM, gradient boosting, ...) produces a predictor $\widehat{g}_{\boldsymbol{\lambda}}^{(n)} : \mathcal{X} \to \mathcal{Y}$. Note that there can be some randomness in the training process (e.g., if stochastic gradient descent is used) so that $\widehat{g}_{\boldsymbol{\lambda}}^{(n)}$ may still be random for a given training set and hyper-parameters. The goal is to build a predictor that generalizes well. If we had access to the distribution $\mathbf{P}$ that generated the data (i.e., assuming that data points in $\mathcal{D}_{\texttt{train}}$ are i.i.d. from $\mathbf{P}$), this generalization power would be measured by the risk $f(\boldsymbol{\lambda}) \triangleq \mathbb{E}\left[\ell\left(\boldsymbol{Y}, \widehat{g}_{\boldsymbol{\lambda}}^{(n)}(\boldsymbol{X})\right)\right]$, where $\ell$ is some loss function measuring the distance between two predictions and the expectation is taken on $(\boldsymbol{X}, \boldsymbol{Y}) \sim \mathbf{P}$ and the possible randomness in the training process.

In practice, however, the explicit evaluation of $f$ is impossible, but there are several methods for *noisy evaluations.* We can either compute the validation error of $\widehat{g}_{\boldsymbol{\lambda}}^{(n)}$ on a held-out validation set, $1/|\mathcal{D}_{\texttt{valid}}| \sum_{i=1}^{|\mathcal{D}_{\texttt{valid}}|} \ell(\widehat{g}_{\boldsymbol{\lambda}}^{(n)}(\mathbf{x}_i), \mathbf{y}_i)$, or a cross validation error over the training set as an approximation of the objective.

## 3. Best-arm identification for HPO

Hyper-parameter optimization can be modeled as a BAI task in a bandit model. Given a finite set of arms $\mathcal{A} \triangleq \{1, \ldots, K\}$, when we select arm $i$, we get an independent observation from some unknown distribution $\nu_i$ with mean $\mu_i$. A BAI algorithm sequentially selects arms in order to identify the arm with the largest mean, $I^\star \triangleq \arg\max_{i \in \mathcal{A}} \mu_i$[2]. In the context of HPO, each arm models the quality of a given hyper-parameter configuration $\boldsymbol{\lambda}$. When the arm is sampled, a noisy evaluation of $f(\boldsymbol{\lambda})$ is received, which is the mean reward of that arm. A BAI (or pure-exploration) algorithm consists of a sequential arm selection strategy, indicating which arm $I_t$ is selected at round $t$, coupled with *recommendation rule* that selects a candidate best arm $I_t^\star$ at round $t$. The goal is either to minimize the error probability $\mathbf{P}(\mu_{I_t^\star} \neq \mu_{I^\star})$ (Audibert and Bubeck, 2010; Karnin et al., 2013) or the *simple regret* (Bubeck et al., 2009; Gabillon et al., 2012), defined as $r_t = \mu_{I^\star} - \mu_{I_t^\star}$, possibly after a total budget $B$, whose knowledge may be used by the algorithm. Note that the BAI problem can also be studied from a fixed-confidence point of view (Even-dar et al., 2003).

2. Here we present BAI problems in a standard way for which we search for an arm with the largest mean. For HPO, however, it is important to mention that we are searching for a hyper-parameter configuration that minimizes the validation error. One can easily see that it does not change the problem in principle.

Standard BAI algorithms are however not straightforwardly applicable to HPO when the search space can be infinite and is often continuous. To handle those interesting cases, we rather turn our attention to BAI in a infinitely many armed bandit (Carpentier and Valko, 2015). In this context, there is an infinite pool of arms, whose means are assumed to be drawn from some *reservoir distribution* $\nu_0$. In such a model, a BAI algorithm maintains a list of arms (hyper-parameter configurations) that have been tried before. At each round it can either query a new arm from the reservoir (add a new hyper-parameter, selected at random, to the current pool), add it to the list and sample it (evaluate it), or sample an arm already in the list (re-evaluate a configuration tried before).

A natural way to perform BAI in an infinite-many armed bandit model consists in first querying a *well-chosen* number of arms from the reservoir and then running a standard BAI algorithm on those arms (Carpentier and Valko, 2015). However this ideal number may rely on the difficult of the learning task, which is hardly known in practice. The `Hyperband` algorithm (Li et al., 2017) takes a step further and successively queries several batches of arms from the reservoir, including a decreasing number of arms in each batch, while increasing the budget dedicated to each of them. `SHA` (Karnin et al., 2013), a state-of-the-art BAI algorithm, is then run on each of these batches of arms. This approach seems more robust in that it trades off between *the number of arms that is needed to capture a good arm* and *how much measurement effort we should allocate to each of them.* However, a numerical study performed by Aziz et al. (2018b) seems to reveal that an infinite bandit algorithm based on `SHA` should always query the maximal number of arms from the reservoir[3].

All the existing algorithms are still subject to a pre-defined scheduling of how many arms should be queried from the reservoir. The algorithm (`D-TTTS`) we proposed in this paper does not need to decide in advance how many arms will be queried, and is therefore fully *dynamic.*

**Remark 1** *`Hyperband` is proposed specifically for hyper-parameter tuning. Its original philosophy is to adaptively allocate* resources *to more promising configurations. Resources here can be time, dataset sub-sampling, feature sub-sampling, etc. In such a setting, the classifier is not always trained into completion given a parameter configuration, but is rather stopped early if it is shown to be bad so that we can allocate more resources to other configurations. In this case, different evaluations of a single configuration cannot be considered as i.i.d. anymore. Thus, HPO is stated as a non-stochastic infinitely-armed bandit problem. This idea of early stopping is also further investigated by combining Bayesian optimization with it (Falkner et al., 2018). However, this is out of the scope of this paper.*

## 4. Active Thompson sampling for HPO

In this section, we introduce a new algorithm for BAI in an infinite bandit model, that is an adaptation of Thompson sampling (Thompson, 1933). Thompson sampling can be seen as the very first bandit algorithm ever proposed, but has been used for the *rewards maximization* objective, which is quite different from BAI, as explained by Bubeck et al. (2009). Instead of using vanilla Thompson sampling, we build on `TTTS` that is an adaptation

3. This reference is a preliminary draft that has been withdrawn due to technical issues in the proofs. Yet we believe the experimental section to be sound.

of Thompson sampling for BAI in finite-arm bandits. Unlike the state-of-the-art algorithm `SequentialHalving` that requires the knowledge of the total budget to operate, `TTTS` is particularly appealing as it does not need to have it. Such algorithms are referred to as *anytime*. Besides, it is known to be optimal in a Bayesian (asymptotic) sense, as it attains the best possible rate of decay of the posterior probability of the set of wrong models.

As a Bayesian algorithm, `TTTS` uses a prior distribution $\Pi_0$ over the vector of means of the $K$ arms, $\boldsymbol{\mu} \triangleq (\mu_1, \ldots, \mu_K)$, which can be updated to a posterior distribution $\Pi_t$ after $t$ observations. Under the Bernoulli bandit model, arm $i$ produces a reward $Y_{t,i} = 1$ with probability $\mu_i$, and $Y_{t,i} = 0$ with probability $1 - \mu_i$ when sampled at round $t$. Given independent uniform prior for the mean of each arm, the posterior distribution on $\boldsymbol{\mu}$ is a product of $K$ Beta distributions: $\Pi_t = \bigotimes_{i=1}^{K} \mathtt{Beta}(1 + S_{t,i}, N_{t,i} - S_{t,i} + 1)$, where $N_{t,i}$ is the number of selections of arm $i$ until round $t$ and $S_{t,i}$ is the sum of rewards obtained from that arm. At each round $t$, `TTTS` chooses one arm from the following two candidates to evaluate: (1) it first samples a parameter $\boldsymbol{\theta}$ from $\Pi_{t-1}$, and the first candidate is defined as $I_t^{(1)} \triangleq \arg\max_{i\in\mathcal{A}} \theta_i$, (2) it repeatedly samples new $\boldsymbol{\theta}'$ until $I_t^{(2)} \triangleq \arg\max_{i\in\mathcal{A}} \theta'_i$ is different from $I_t^{(1)}$. `TTTS` depends on a parameter $\beta \in (0, 1)$. In particular, the algorithm selects $I_t = I_t^{(1)}$ with probability $\beta$ and $I_t = I_t^{(2)}$ with probability $1 - \beta$.

`TTTS` can also be used for bandit settings in which the rewards are bounded in $[0, 1]$ by using a binarization trick first proposed by Agrawal and Goyal (2012): When a reward $Y_{t,i} \in [0, 1]$ is observed, the algorithm is updated with a fake reward $Y'_{t,i} \sim \mathtt{Ber}(Y_{t,i}) \in \{0, 1\}$. `TTTS` can thus be used for BAI for a *finite* number of arms that with rewards in $[0, 1]$. We now present a simple way of extending `TTTS` to deal with an infinite number of arms.

**Dynamic `TTTS`** In an infinite bandit algorithm, at each round, we either query a new arm from the reservoir *and sample it*, or re-sample a previous arm. In a Bayesian setting, we can also imagine that at each round, an arm is queried from the reservoir and added with a *uniform prior* to the list of queried arms, *regardless of whether it is sampled or not*. Then, at round $t$, `D-TTTS` consists in running `TTTS` on these $t$ arms, out of which several are endowed with a uniform prior and have never been sampled.

Leveraging the fact the the maximum of $k$ uniform distribution has a $\mathtt{Beta}(k, 1)$ distribution and that `TTTS` only depends on the maxima of posterior samples, we give the following equivalent implementation for `D-TTTS` (Algorithm 1). Letting $\mathcal{L}_t$ be the list of arms that have been queried from the reservoir and sampled *at least once* before round $t$, at round $t$ we run `TTTS` on the set $\mathcal{A}_t \triangleq \mathcal{L}_t \cup \{\mu_0\}$ where $\mu_0$ is a pseudo-arm with posterior distribution $\mathtt{Beta}(t - k_t, 1)$, where $k_t \triangleq |\mathcal{L}_t|$.

It remains to decide how to recommend the arm as our best guess. In this paper, we choose the most natural recommendation strategy for Bayesian algorithms that outputs the arm with the largest *posterior probability of being optimal*. Letting $\Theta_i$ be the subset of the set $\Theta$ of possible mean vectors such that arm $i$ is optimal, $\Theta_i \triangleq \left\{\boldsymbol{\theta} \in \Theta \,|\, \theta_i > \max_{j\neq i} \theta_j\right\}$, the posterior probability that arm $i$ is optimal after round $t$ is defined as $\Pi_t(\Theta_i)$. At any time $t$, we therefore recommend arm $\widehat{I}_t \triangleq \arg\max_{i\in\mathcal{A}} \Pi_t(\Theta_i)$.

**Hyper `TTTS`** We briefly present another way of extending `TTTS` to deal with an infinite number of arms, namely `Hyper-TTTS` or `H-TTTS` that we use as a baseline in the experiments. `H-TTTS` is a variant of `Hyperband` in which `SHA` is replaced by `TTTS`. This algorithm, whose

**Algorithm 1** Sampling rule of Dynamic TTTS (D-TTTS)

```
Input: $\beta$; $B$ (total budget); $\nu_0$
Initialization: $\mu_1 \sim \nu_0$; $t \leftarrow 0$; $\mathcal{A} \leftarrow \{\mu_0, \mu_1\}$; $m \leftarrow 1$; $S_0, N_0 \leftarrow 0$; $S_1 \sim \texttt{Ber}(\mu_1)$, $N_1 \leftarrow 1$
1:  while $t < B$ do
2:    $\forall i = 0, \dots, m$, $\theta_i \sim \texttt{Beta}(S_i + 1, N_i - S_i + 1)$; $U \sim \mathcal{U}([0,1])$
3:    $I^{(1)} \leftarrow \arg\max_{i=0,\dots,m} \theta_i$
4:    if $U > \beta$ then
5:      while $I^{(2)} \neq I^{(1)}$ do
6:        $\forall i = 0, \dots, m, \theta'_i \sim \texttt{Beta}(S_i + 1, N_i - S_i + 1)$
7:        $I^{(2)} \leftarrow \arg\max_{i=0,\dots,m} \theta'_i$
8:      end while
9:      $I^{(1)} \leftarrow I^{(2)}$
10:   end if
11:   if $I^{(1)} \neq 0$ then
12:     $Y \leftarrow$ evaluate arm $I^{(1)}$; $X \sim \texttt{Ber}(Y)$
13:     $S_{I^{(1)}} \leftarrow S_{I^{(1)}} + X$; $N_{I^{(1)}} \leftarrow N_{I^{(1)}} + 1$; $S_0 \leftarrow S_0 + 1$
14:   else
15:     $\mu_{m+1} \sim \nu_0$; $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mu_{m+1}\}$;
16:     $Y \leftarrow$ evaluate arm $m+1$; $X \sim \texttt{Ber}(Y)$
17:     $S_{m+1} \leftarrow X$; $N_{m+1} \leftarrow 1$; $m \leftarrow m + 1$
18:   end if
19:   $t \leftarrow t + 1$
20: end while
```

sampling rule is formally stated as Algorithm 2, runs $s_{\max}$ batches of TTTS with different number of arms $n$ and each batch with a same budget $T = \lceil B/s_{\max} \rceil$ with $B$ the total budget. The number of arms within each bracket is decreasing with an exponential rate of $\gamma$. One inconvenience of this algorithm is that $s_{\max}$ and $\gamma$ still need to be tuned (in practice, we use the same tuning as the one of Hyperband).

**Some synthetic results** Now we give some synthetic experimental results comparing D-TTTS to Hyperband and to SHA, a state-of-the-art algorithm for finite BAI that can be adapted to the infinite setting. In those experiments, the arms are Bernoulli distributed and the reservoir distribution $\nu_0$ is fixed to some Beta$(a, b)$ distribution.

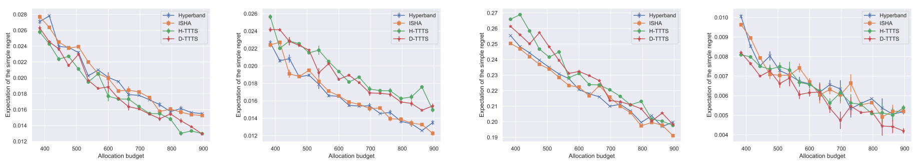
Infinite Sequential Halving (ISHA) consists in running SHA on a fixed number of arms drawn from the reservoir. Observe that for a total budget $B$, there exists a maximum number of arms $K^\star$ that can be processed by SHA, which satisfies $B = \lceil K^\star \log_2(K^\star) \rceil$. Following Aziz et al. (2018b), we run ISHA with $K^\star$ arms drawn from the reservoir.

We report in Fig. 1 the simple regret as a function of time for different algorithms and four Beta reservoir distributions. H-TTTS and D-TTTS are run with $\beta = 1/2$ which is known to be a robust choice (Russo, 2016). Each point represents the expected simple regret $\mathbb{E}[1 - \mu_{I_n^*}]$ estimated over 1000 replications for an algorithm run with budget $n$. D-TTTS is very competitive on 3 reservoirs and H-TTTS is sometimes better, sometimes worse than Hyperband. We also tried the SiRI algorithm (Carpentier and Valko, 2015) (with $b$ as the

**Algorithm 2** Sampling rule of Hyper `TTTS` (`H-TTTS`)

```
Input: $\beta$; $\gamma$; $B$; $s_{\max}$; $\nu_0$
Initialization: $T = \lfloor B/s_{\max} \rfloor$
1:  for $s \leftarrow s_{\max}$ to 0 do
2:    $K = \left\lceil \frac{s_{\max}+1}{s+1}\gamma^s \right\rceil$
3:    $\mathcal{A} \leftarrow \{i = 1, \dots, K : \mu_i \sim \nu_0\}$; $t = 0$
4:    while $t < T$ do
5:      sample $\boldsymbol{\theta} \sim \Pi_t$
6:      $I^{(1)} \leftarrow \arg\max_{i\in\mathcal{A}} \theta_i$
7:      sample $b \sim \text{Bernoulli}(\beta)$
8:      if $b = 1$ then
9:        $Y \leftarrow$ evaluate arm $I^{(1)}$
10:     else
11:       while $I^{(2)} \neq I^{(1)}$ do
12:         $\forall i \in \mathcal{A}, \theta'_i \sim \texttt{Beta}(S_i + 1, N_i - S_i + 1)$
13:         $I^{(2)} \leftarrow \arg\max_{i\in\mathcal{A}} \theta'_i$
14:       end while
15:       $I^{(1)} \leftarrow I^{(2)}$
16:       $Y \leftarrow$ evaluate arm $I^{(1)}$
17:     end if
18:     $X \sim \texttt{Bernoulli}(Y)$
19:     $S_{I^{(1)}} \leftarrow S_{I^{(1)}} + X$; $N_{I^{(1)}} \leftarrow N_{I^{(1)}} + 1$
20:     $t = t + 1$
21:   end while
22: end for
```

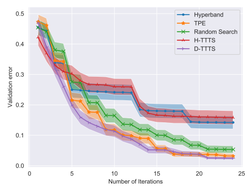
tail parameter when $\nu_0$ =Beta$(a, b)$) but obtained worse performance and therefore do not report the results.



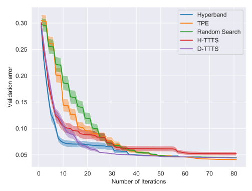
(a) Beta(1,1) (b) Beta(3,1) (c) Beta(1,3) (d) Beta(0.5,0.5)

Figure 1: expected simple regret as a function of the number of arms evaluations for different Beta reservoir

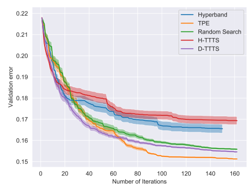
Note that in the implementation of `Hyperband` for this *stochastic* infinite bandit setting, the elimination phase of the underlying `SHA` algorithm is carried out according to the averaged loss of previous samples (as samples from an arm are i.i.d. in this setting and not a converging sequence). In the next section, we apply our algorithm to some real hyper-parameter optimization tasks.



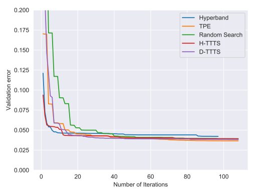
(a) wine



(b) breast cancer



(c) adult



(d) `MNIST`

## 5. Experiments

We benchmark our bandit-based strategy against different types of HPO algorithms, namely, `TPE`, random search, `Hyperband` and `H-TTTS`, for the tuning of classifiers (`SVM` and `MLP`) on 4 different classification tasks: *wine*, *breast cancer*, and *adult* datasets from `UCI` machine learning repository (Dua and Taniskidou, 2017); and the `MNIST` dataset (LeCun et al., 1998).

For all the methods, a noisy evaluation of the black-box function $f$ (see the terminology introduced in Section 2) for a hyper-parameter configuration $\boldsymbol{\lambda}$ consists in performing a shuffled 3-fold cross-validation on $\mathcal{D}_{\texttt{train}}$. More precisely, given a random partitioning $\cup_{j=1}^{3}\mathcal{D}_{\texttt{valid}}^{j}$ of $\mathcal{D}_{\text{train}}$, where the folds are of equal size, we train a classifier $\widehat{g}_{\boldsymbol{\lambda}}^{(j)}$ on $\mathcal{D}_{\texttt{train}}\backslash\mathcal{D}_{\texttt{valid}}^{j}$ for each fold $j$ and compute the average validation error defined as $e \triangleq 1/|\mathcal{D}_{\texttt{train}}|\sum_{j=1}^{3}\sum_{i\in\mathcal{D}_{\text{valid}}^{j}}\mathbb{1}\{\widehat{g}_{\boldsymbol{\lambda}}^{(j)}(\mathbf{x}_i) \neq \mathbf{y}_i\}$, which we report as a noisy estimate of the risk $f(\boldsymbol{\lambda}) \triangleq \mathbf{P}(\widehat{g}_{\boldsymbol{\lambda}}^{(n)}(\boldsymbol{X}) \neq \boldsymbol{Y})$.

Observe that both the noisy evaluation and the value of $f$ belong to $[0,1]$. Therefore we can introduce an *arm* with rewards in $[0,1]$ for each hyper-parameter $\boldsymbol{\lambda}$. Sampling arm $\boldsymbol{\lambda}$ produces reward $r \triangleq 1-e \in [0,1]$ with a different random partitioning and random seed for training for each selection. Arm $\lambda$ is assumed to have mean of $1-f(\boldsymbol{\lambda})$. In an infinite arm setting, querying a new arm from the reservoir corresponds to selecting a new hyper-parameter at random from the search space. With these two notions (*arm sampling* and *reservoir querying*), our algorithm for infinite BAI applies to HPO.

For the experiments, we adapt the recommendation rule of `D-TTTS` to the HPO applications considered and always recommend the hyper-parameter configuration that has produced the smallest cross-validation error so far (which is also the recommendation rule used by other approaches, e.g., `Hyperband`). For all methods, we report the cross-validation error for the recommended hyper-parameter configuration, as a function of time. We stress again that, unlike in standard bandits, where we could use the simple regret as a performance metric, we do not have access to the ground truth generalization error in real classification tasks. Therefore, we only report a proxy of the true error rate that we are interested in.

**Results** We first benchmark[4] our methods on a few simple `UCI` datasets using `SVM` from `scikit-learn` as the classifier. We optimize over two hyper-parameters: the *penalty parameter* $C$ and the *kernel coefficient* $\gamma$[5] for an RBF kernel, for which the pre-defined search bounds are both $\left[10^{-5}, 10^{5}\right]$.

Fig. 2a shows the mean cross-validation error of `SVM` run on the `UCI` wine dataset over 24 pulls[6] averaged on 100 runs. The task is to predict the quality score of wine (between

4. code at `http://researchers.lille.inria.fr/~valko/hp/publications/shang2019simple.code.zip`
5. $\gamma$ is the parameter of the RBF kernel defined as $\exp(-\gamma||\mathbf{x}-\mathbf{x}'||^2)$
6. the number of pulls here and later is chosen exactly as in the work of Li et al. (2017)

0 and 10) given 11 attributes. Recall that one iteration corresponds to one arm pull. In this experiment, D-TTTS improves over other benchmark algorithms. Fig. 2b is the same experiment run on the UCI breast cancer dataset over 81 pulls. The task is to predict whether a patient has breast cancer based on 32 attributes. We repeat the experiment 100 times. This time, D-TTTS is slightly worse than Hyperband at the beginning, but improves later. Finally, we optimize SVM on a relatively more complicated UCI adult dataset over 162 pulls, for which the result is shown in Fig. 2c. The task is to tell whether the income of an individual is higher than 50k or not given 14 attributes. This experiment is also averaged over 100 runs. D-TTTS is better than other algorithms at the beginning, but is outperformed by TPE towards the end. We see that, although not always the best, D-TTTS shows a consistent, robust, and quite competitive performance in the 3 tasks.

We now carry out the classic MNIST digits classification task using multi-layer perceptron (MLP). We choose to optimize over three hyper-parameters: the *size of hidden layer* (an integer between 5 and 50), the $\ell_2$ *penalty parameter* $\alpha$ (between 0 and 0.9) and the *initial learning rate* (bounded in $\left[10^{-5}, 10^{-1}\right]$). Fig. 2d shows the result of MLP run on MNIST over 108 pulls, this time averaged over 20 runs. D-TTTS is slightly worse than Hyperband and H-TTTS in the very beginning, but is performing well afterward.

## 6. Adaptivity to $\mu^\star$

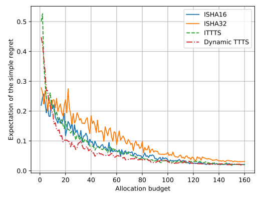
In this section we provide more insights on D-TTTS. In particular, we discuss a main drawback of the current version of D-TTTS by further experimental illustrations and propose an overcome afterward.

### 6.1 Illustration of effectively sampled arms by D-TTTS

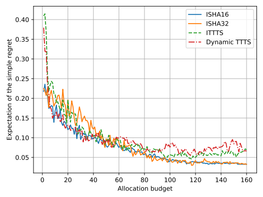
One drawback of the current D-TTTS is that it may not work well if we do not know the oracle $\mu^\star$ ($\mu^\star$ is set to 1 in our previous experiments). Fig. 3 shows the expected simple regret of D-TTTS compared to ISHA and TTTS under a $\texttt{Beta}(0.5, 0.5)$ reservoir shifted by 0.8, 0.6, 0.4, 0.2 and without shift respectively. A Beta distribution $\texttt{Beta}(a, b)$ shifted by $\mu^\star$ is obtained by re-scaling to $[0, \mu^*]$ the corresponding distribution. More formally, a shifted Beta distribution on $[0, \mu^*]$, denoted by $\texttt{SB}_{\mu^\star}(a, b)$ in the rest of the paper, is the distribution of $X\mu^*$ where $X \sim \texttt{Beta}(a, b)$ (see Section 6.2 for more discussion on shifted Beta distributions). We can see that the performance of D-TTTS is getting worse along with the increasing shift.

As suggested by the implementation trick introduced in Section 4, all the $k$ arms that have been added but not effectively sampled can be seen as a virtual arm endowed with a $\texttt{Beta}(k, 1)$ posterior. Intuitively, if $\mu^\star < 1$, than this virtual arm would force the algorithm to sample too many new arms, thus would lack of attention on arms that are more likely to be near-optimal. This intuition is supported by the illustration in Fig. 4a: the posterior distributions of effectively sampled will eventually be supported mostly on the left of $\mu^*$, while the pseudo-arm still put a lot of mass near 1.

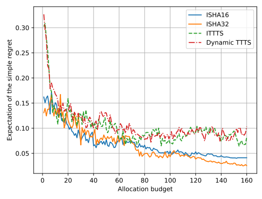
In Fig. 4b, we report the number of arms that have been played $1, 2, \ldots, 9$ and more than 10 times for D-TTTS run under a $\texttt{Beta}(0.5, 0.5)$, $\texttt{SB}_{0.8}(0.5, 0.5)$, $\texttt{SB}_{0.6}(0.5, 0.5)$, $\texttt{SB}_{0.4}(0.5, 0.5)$ and $\texttt{SB}_{0.2}(0.5, 0.5)$ reservoir respectively, which confirms the over-exploration effect caused by shifted reservoirs.



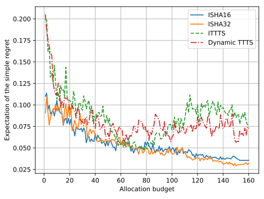
(a) no shift



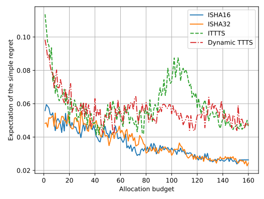
(b) shift by 0.8



(c) shift by 0.6

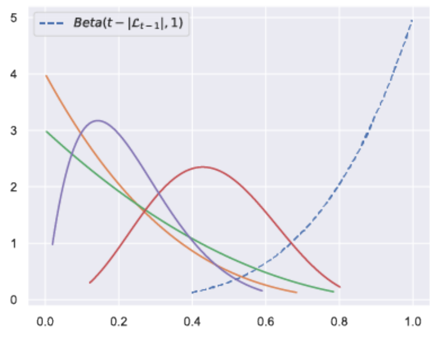


(d) shift by 0.4

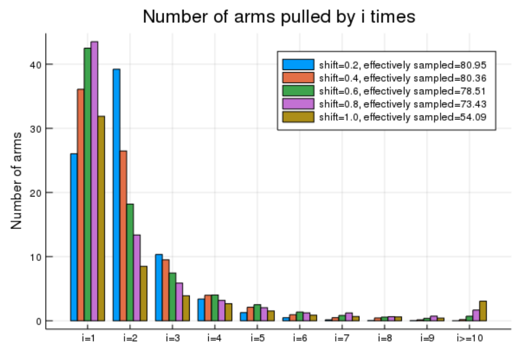


(e) shift by 0.2

Figure 3: expected simple regret for shifted Beta reservoir, averaged over 100 runs



(a) posterior distributions of the effectively samples arms and the pseudo-arm



(b) number of effectively sampled arms, averaged over 100 runs

Figure 4: illustration of over-exploration under shifted reservoirs

In the next section, we propose a simple fix for the algorithm that is not tailored for $\mu^* = 1$ but adjusts to any value of $\mu^*$.

### 6.2 A simple adaptive fix

We now propose a natural extension of D-TTTS to overcome the issue mentioned in the previous section. In this section we assume that we have the knowledge of the maximum mean $\mu^\star$ of the reservoir. The core idea is to keep the same algorithm but with a different prior distribution over each queried arm, that is supported on $[0, \mu^\star]$ instead of $[0, 1]$.

**Bernoulli bandits** We still assume a Bernoulli bandit model for the rewards (although the algorithm is extended to any rewards bounded in $[0, 1]$ with the binarization trick):

An arbitrary arm produces at time $t$ a reward 1 with probability $\theta$ and a reward 0 with probability $1-\theta$. The likelihood can be written as follow:

$$p(s|\theta) = \theta^s (1-\theta)^{1-s}; s \in \{0;1\}.$$

**Sample from the shifted posterior** In order to implement the extension of D-TTTS, we need to know how to sample from the "shifted" posterior, that is the posterior assuming a uniform prior over $[0, \mu^*]$ instead of $[0,1]$. We now explain how to compute this posterior distribution on $\theta$ given a sequence of observations $Y_1, Y_2, \cdots, Y_N \in \{0;1\}$. Define

$$\begin{cases} a & = \sum_{i=1}^N Y_i + 1 \\ b & = N - \sum_{i=1}^N Y_i + 1, \end{cases}$$

then, according to the Bayes rule, we have

$$\begin{aligned} p(\theta|Y_1,\cdots,Y_N) &= \frac{p(Y_1,\cdots,Y_N|\theta)p(\theta)}{p(Y_1,\cdots,Y_N)} \\ &= \frac{p(Y_1,\cdots,Y_N|\theta)p(\theta)}{\int_0^1 p(Y_1,\cdots,Y_N|\theta')p(\theta')\mathbb{1}_{[0,\mu^\star]}(\theta')d\theta'} \\ &= \frac{\theta^{a-1}(1-\theta)^{b-1}\mathbb{1}_{[0,\mu^\star]}(\theta)/B(a,b)}{\int_0^{\mu^\star}(\theta')^{a-1}(1-\theta')^{b-1}/B(a,b)d\theta'} \\ &= \frac{\theta^{a-1}(1-\theta)^{b-1}\mathbb{1}_{[0,\mu^\star]}(\theta)}{B(a,b)F_{a,b}(\mu^\star)}, \end{aligned}$$

where $F_{a,b}$ is the *cumulative distribution function* (cdf) of $\mathtt{Beta}(a,b)$. Thus the cdf of the posterior is

$$\mathbb{P}\left[\theta \le x | Y_1,\cdots,Y_N\right] = \frac{F_{a,b}(x)}{F_{a,b}(\mu^\star)} \triangleq G(x).$$

Now the sampling is quite straightforward as $G^{-1}(u)$ can be computed as

$$G^{-1}(u) = F_{a,b}^{-1}(u * F_{a,b}(\mu^\star)).$$

The computation of $F_{a,b}^{-1}$ and $F_{a,b}$ is easily accessible via existing libraries in different programming languages, and we can thus apply *inverse transform sampling* to obtain the observations, since if $U \sim \mathcal{U}([0,1])$, then $G^{-1}(U)$ follows the posterior distribution.

**Shifted Beta distribution** Recall that we defined a shifted Beta distribution $\mathtt{SB}_{\mu^\star}(a,b)$ as the distribution of the random variable $\theta' \triangleq \mu^\star\theta$, where $a, b$ are the shape hyper-parameters of the Beta distribution and $\theta \sim \mathtt{Beta}(a,b)$. The *probability density function* (pdf) of $\mathtt{SB}_{\mu^\star}(a,b)$ can be written as

$$p(\theta') = \frac{1}{B(a,b)} \frac{(\theta')^{a-1}(\mu^\star - \theta')^{b-1}}{(\mu^\star)^{a+b-1}},$$

via the transformation $\theta' = \mu^\star\theta$. Here $B$ is the Beta function[7].

The previous expression is particularly useful if we want to use the same efficient implementation trick that we employed in Algorithm 1, namely the order statistic trick.

7. $B(a,b) \triangleq \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$, and $\Gamma$ is the Gamma function

**Order statistic trick** Now we show that an "order statistic trick" still exists under a uniform prior over $[0, \mu^*]$, namely that the maximum of $k$ random variables drawn from this prior distribution still has a nice distribution.

Given $n$ random variables $X_1, X_2, \cdots, X_n$, the order statistics $X_{(1)}, X_{(2)}, \cdots, X_{(n)}$ are also random variables, defined by sorting the values of $X_1, X_2, \cdots, X_n$ in an increasing order. In this section we treat the special case where they are i.i.d samples from the same distribution with a cdf. $F_X$. Following Gentle (2009), Chapter 1 Section 7, we know that the cumulative distribution function of the $k$-th order statistic can be written as follow:

$$F_{X_{(k)}}(x) = \sum_{j=k}^{n} (F_X(x))^j (F_X(x))^{n-j}.$$

Now, in our case, where the underlying distribution is the uniform distribution defined over $[0, \mu^\star]$, we obtain the pdf of the order statistic $X_{(k)}$ as follow:

$$\begin{aligned} p_{X_{(k)}}(\theta') &= \frac{n!}{(k-1)!(n-k)!} (\mu^\star)^n (\theta')^{k-1} (\mu^\star - \theta')^{n-k} \\ &= \frac{1}{B(k, n+1-k)} \frac{(\theta')^{k-1} (\mu^\star - \theta')^{n-k}}{(\mu^\star)^{(k-1)+(n-k)+1}}. \end{aligned}$$

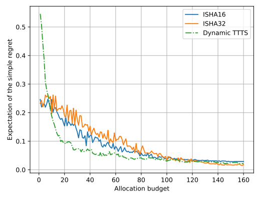
We recognize the density of a shifted Beta distribution with $k$ and $n+1-k$ as shape hyper-parameters. In particular, in our case, the pseudo arm at time $t$ is endowed with the distribution $\mathtt{SB}_{\mu^\star}(t - k_t, 1)$.

**Some illustrations of the fix** Now we show some synthetic results after the previous tricks. Fig. 5 shows the expected simple regret of D-TTTS compared to ISHA, again, under Beta$(0.5, 0.5)$, $\mathtt{SB}_{0.8}(0.5, 0.5)$, $\mathtt{SB}_{0.6}(0.5, 0.5)$, $\mathtt{SB}_{0.4}(0.5, 0.5)$ and $\mathtt{SB}_{0.2}(0.5, 0.5)$ reservoir respectively. We can see that the performance of D-TTTS for shifted cases has been significantly enhanced.

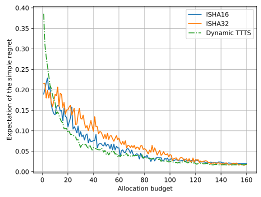
We can also compare the number of effectively sampled arms under shifted Beta reservoirs before and after the fix, as shown in Fig. 6. Fig. 6a is the same figure as Fig. 4b, and Fig. 6b is the number of effectively sampled arms after the previous fix under a Beta$(0.5, 0.5)$, $\mathtt{SB}_{0.8}(0.5, 0.5)$, $\mathtt{SB}_{0.6}(0.5, 0.5)$, $\mathtt{SB}_{0.4}(0.5, 0.5)$ and $\mathtt{SB}_{0.2}(0.5, 0.5)$ reservoir respectively. Indeed, we can see that now the exploration effort of D-TTTS under shifted Beta priors is more or less at the same level as that under a normal Beta reservoir.

## 7. Discussion

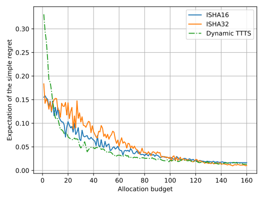
We presented a way to use Thompson sampling for BAI for infinitely many-armed bandits and explained how to use it for HPO. We introduced the *first fully dynamic algorithm* for this setting and showed through an empirical study that it is a promising approach for HPO. In the future, we plan to provide experiments on more datasets and establish theoretical guarantees to support the good performance of D-TTTS, with the hope to provide a finite-time upper bound on its probability of error. We also plan to investigate variants of this algorithm for the non-stochastic bandits for which Hyperband can be used, which would allow spending more time on the more promising algorithms.



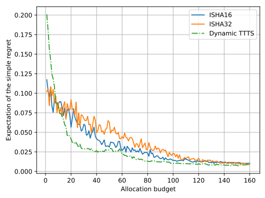
(a) no shift



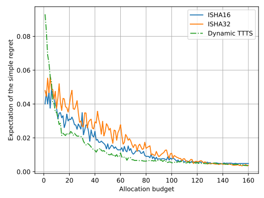
(b) shift by 0.8



(c) shift by 0.6

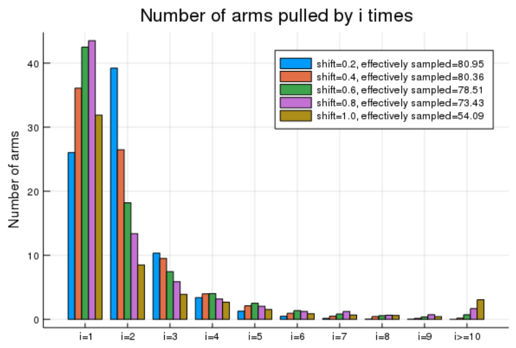


(d) shift by 0.4

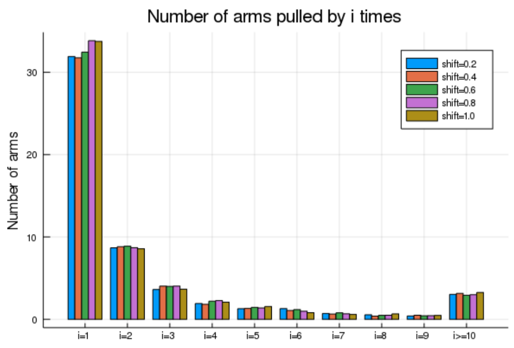


(e) shift by 0.2

Figure 5: expected simple regret for shifted Beta reservoir after the fix, averaged over 100 runs



(a) shift



(b) shift after fix

Figure 6: distribution of effectively sampled arms before and after the fix, averaged over 100 runs

## References

Shipra Agrawal and Navin Goyal. Analysis of Thompson sampling for the multi-armed bandit problem. In *Proceedings of the 25th Conference on Learning Theory (CoLT)*, pages 1–26, 2012.

Jean-Yves Audibert and Sébastien Bubeck. Best arm identification in multi-armed bandits. In *Proceedings of the 23rd Conference on Learning Theory (CoLT)*, 2010.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multi- armed bandit problem. *Machine Learning Journal*, 47(23):235–256, 2002.

Maryam Aziz, Jesse Anderton, Emilie Kaufmann, and Javed Aslam. Pure exploration in infinitely-armed bandit models with fixed-confidence. In *Proceedings of the 29th International Conference on Algorithmic Learning Theory (ALT)*, 2018a.

Maryam Aziz, Kevin Jamieson, and Javed Aslam. Pure-exploration for infinite-armed bandits with general arm reservoirs. *arXiv preprint arXiv:1811.06149*, 2018b.

Peter L. Bartlett, Victor Gabillon, and Michal Valko. A simple parameter-free and adaptive approach to optimization under a minimal local smoothness assumption. In *Proceedings of the 30th International Conference on Algorithmic Learning Theory (ALT)*, 2019.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24 (NIPS)*, pages 2546–2554, 2011.

Donald A. Berry, Robert W. Chen, Alan Zame, David C. Heath, and Larry A. Shepp. Bandit problems with infinitely many arms. *Annals of Statistics*, 25(5):2103–2116, 1997.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in multi-armed bandits problems. In *Proceedings of the 20th International Conference on Algorithmic Learning Theory (ALT)*, pages 23–37, 2009.

Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvari. X-armed bandits. *Journal of Machine Learning Research*, 12:1587–1627, 2010.

Alexandra Carpentier and Michal Valko. Simple regret for infinitely many armed bandits. In *Proceedings of the 32nd International conference on Machine Learning (ICML)*, pages 1133–1141, 2015.

Dheeru Dua and Karra E. Taniskidou. UCI Machine Learning Repository, 2017.

Eyal Even-dar, Shie Mannor, and Yishay Mansour. Action elimination and stopping conditions for reinforcement learning. In *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 162–169, 2003.

Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 2018.

Victor Gabillon, Mohammad Ghavamzadeh, and Alessandro Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Advances in Neural Information Processing Systems 25 (NIPS)*, pages 3212–3220, 2012.

James E. Gentle. *Computational Statistics*. Springer-Verlag New York, 2009.

Jean-Bastien Grill, Michal Valko, and Rémi Munos. Black-box optimization of noisy functions with unknown smoothness. In *Advances in Neural Information Processing Systems 28 (NIPS)*, pages 667–675, 2015.

Matthew W. Hoffman, Bobak Shahriari, and Nando de Freitas. On correlation and budget constraints in model-based bandit optimization with application to automatic machine learning. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AIStats)*, pages 365–374, 2014.

Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION)*, pages 507–523, 2011.

Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *Proceedings of the 30th International Conference on Machine Learning (ICML)*, pages 1238–1246, 2013.

Aaron Klein, Stefan Falkner, Numair Mansur, and Frank Hutter. RoBO: A flexible and robust Bayesian optimization framework in Python. In *7th Workshop on Bayesian Optimization at Neural Information Processing Systems (NIPS-BayesOpt)*, 2017.

Nicolas Knudde, Joachim van der Herten, Tom Dhaene, and Ivo Couckuyt. GPflowOpt: A Bayesian optimization library using TensorFlow. *arXiv preprint arXiv:1711.03845*, 2017.

Tor Lattimore and Csaba Szepesvari. *Bandit Algorithms*. Cambridge University Press, 2018.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Ameet Talwalkar, and Afshin Rostamizadeh. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proceedings of the 5th International Conference on Learning Representations (ICLR)*, 2017.

Ilya Loshchilov and Frank Hutter. CMA-ES for hyperparameter optimization of deep neural networks. In *Workshop Track of the 4th International Conference on Learning Representations*, 2016.

Daniel Russo. Simple Bayesian algorithms for best arm identification. In *Proceedings of the 29th Conference on Learning Theory (CoLT)*, 2016.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

Xuedong Shang, Emilie Kaufmann, and Michal Valko. General parallel optimization without a metric. In *Proceedings of the 30th International Conference on Algorithmic Learning Theory (ALT)*, 2019.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25 (NIPS)*, pages 2951–2959, 2012.

Niranjan Srinivas, Andreas Krause, Sham M. Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International conference on Machine Learning (ICML)*, pages 1015–1022, 2010.

William R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285, 1933.

Yizao Wang, Jean-Yves Audibert, and Rémi Munos. Algorithms for infinitely many-armed bandits. In *Advances in Neural Information Processing Systems 21 (NIPS)*, pages 1729–1736, 2008.