Best arm identification for hyper-parameter optimization

Authors EMAIL

Abstract

1. Introduction

Modern machine learning algorithms often contain many nuisance parameters that cannot be learned through the learning process, but instead, need to be manually specified. It is thus appealing to design algorithms that require fewer such so-called *hyper-parameters* (not always feasible though).

An alternative is to perform the *hyper-parameter optimization* (HPO) that can be viewed as a *black-box/global optimization* problem where function evaluations are supposed to be very expensive. Here a typical function evaluation involves running the primary machine learning algorithm to completion on a large and high-dimensional dataset, which often takes a considerable amount of time or resources. This limits vastly the number of evaluations that could be carried out, which makes it desirable to design efficient high-level algorithms that automate this tuning procedure.

Related literature Several naïve but traditional ways of performing the hyper-parameter search exist, such as `GridSearch` and `RandomSearch`.

`GridSearch` is an old-fashioned but commonly used method in a lot of research papers before 2010s (LeCun et al., 1998), as well as in many machine learning softwares such as `LIBSVM` (Chang and Lin, 2011) and `scikit-learn` (Pedregosa et al., 2011). It simply carries out an *exhaustive* searching of parameters through a manually specified *finite* subset of the hyper-parameter search space. `GridSearch` clearly suffers from the *curse of dimensionality*, which makes it undesirable in real applications.

`RandomSearch` overcomes this problem by randomly picking hyper-paramter configurations from the search space, and can be easily generalized to continuous space as well. It is shown to outperform `GridSearch` especially in the case that the *intrinsic dimensionality* of the optimization problem is low (Bergstra and Bengio, 2012). Hence we use `RandomSearch` as a baseline method here in this section. Note that both methods are *embarrasingly parallel*, which can be a very strong point in some situations.

Recent trending solutions for HPO are mostly Bayesian-based (Bergstra et al., 2011; Hutter et al., 2011; Snoek et al., 2012, 2015). *Bayesian optimization* (BO) depends on a prior belief, typically a Gaussian process (GP), on the target function that we can update to a posterior distribution over the function space where lies the target function given a sequence of observations. It then decides where to sample next with the help of an *utility/acquisition function* by maximizing/minimizing the utilitly w.r.t. the posterior distribution. Typical acquisition functions include `EI`, `PI` (Mockus et al., 1978) and `GP-UCB` (Srinivas et al., 2010) (see Brochu et al. 2010; Shahriari et al. 2016 for a survey). Note that classical BO methods with GP prior and typical acquisition functions can be applied straightforwardly to hyper-parameter tuning (Snoek et al. 2012 provides a Python package called `Spearmint` to perform

the task), some variants like `TPE` (Bergstra et al., 2011) and `SMAC` (Hutter et al., 2011) are more commonly used though.

Remark 1 *One may notice that a new optimization task emerges in a Bayesian optimization procedure, that is to optimize the acquisition function. It may seem to be a bit artificial, but this acquisition function is usually much more regular compared to the target function, thus easier to optimize.*

Bayesian optimization focuses on adaptively choosing different parameter configurations based on previous observations, but always run the primary machine learning classifier/regressor into completion given a set of hyper-parameters. In a bandit point of view, this setting can be considered as a *stochastic infinitely-armed bandit* (SIAB) problem.

Another take on this problem is to adaptively allocate *resources* to more promising configurations. Resources here can be time, dataset subsampling, feature subsampling, etc. In such a setting, the classifier is not always trained into completion given a parameter configuration, but rather is stopped early if it is shown to be bad so that we can allocate more resources to other configurations. This idea of early stopping is proposed by Li et al. (2016), which states the HPO problem as a *non-stochastic infinitely-armed bandit* (NIAB) problem.

In this report, we treat both the two settings mentioned previously. Before that, we need to mention that totally different types of approaches also exist, for example *evolutionary optimization* (EO). EO 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.

Finally, hierarchical bandit algorithms like `HOO` (Bubeck et al., 2010), `HCT` (Azar et al., 2014) could be potential candidates for HPO as well. And to the best of our knowledge, these methods have never been investigated in the HPO literature.

Outline We first introduce the general HPO framework in Section 2. Then we present our new heuristic in Section 3 before listing some experimental results in Section 5.

2. Hyper-parameter optimization framework

HPO is known as a *bilevel optimization* problem where one optimization problem is nested in another (Franceschi et al., 2018; Colson et al., 2007). For HPO, the *inner level* task is the classical parameter optimization, while the *outer level* task is to seek a good configuration for the optimized learning algorithm so that it generalizes well. Formally speaking, we are interested in solving an optimization problem of the form

$$\min\{f(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \Omega\},$$

where f is defined as

$$f : \left| \begin{array}{ccc} \Omega & \longrightarrow & \mathbb{R} \\ \boldsymbol{\lambda} & \longmapsto & \inf\{\mathcal{H}(\boldsymbol{\theta}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda}) : \boldsymbol{\theta}_{\boldsymbol{\lambda}} \in \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\theta})\}. \end{array} \right.$$

Here $\mathcal{H} : \mathbb{R}^d \times \Omega \longrightarrow \mathbb{R}$ is called the *outer objective* and $\mathcal{L}_{\boldsymbol{\lambda}} : \mathbb{R}^d \longrightarrow \mathbb{R}$ is the *inner objective*.

Given a set of hyper-parameters $\boldsymbol{\lambda}$, the objective of a (supervised) learning process is to minimize the empirical loss of a machine learning model parameterized by a vector of parameters $\boldsymbol{\theta}$, $g_{\boldsymbol{\theta}} : \mathcal{X} \longrightarrow \mathcal{Y}$. Hence a typical inner objective for a HPO problem is the total empirical loss (could be regularized) over a training set $\mathcal{D}_{\text{train}} = \{(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\dots,|\mathcal{D}_{\text{train}}|}\}$ where $(\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{X} \times \mathcal{Y}$ are pairs of input and output,

$$\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \triangleq \sum_{i=1}^{|\mathcal{D}_{\text{train}}|} \ell_1(g_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i),$$

where the loss function ℓ_1 should be neatly chosen for a specific task.

In the context of hyper-parameter tuning, a typical outer objective is to minimize the validation error of the model $g_{\boldsymbol{\theta}}$, that is to say, we want to minimize the average loss over a holdout validation set $\mathcal{D}_{\text{valid}} = \{(\mathbf{x}_i, \mathbf{y}_i)_{i=1,\dots,|\mathcal{D}_{\text{valid}}|}\}$ (or alternatively a cross validation error over the training set),

$$\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \triangleq \frac{1}{|\mathcal{D}_{\text{valid}}|} \sum_{i=1}^{|\mathcal{D}_{\text{valid}}|} \ell_2(g_{\boldsymbol{\theta}}(\mathbf{x}_i), \mathbf{y}_i)^{1}.$$

In this report, we consider a simplified formalism by assuming an unique maximizer of the inner objective, say for a given $\boldsymbol{\lambda}$, there exists an unique $\boldsymbol{\theta}_{\boldsymbol{\lambda}} = \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\theta})$, then $f(\boldsymbol{\lambda}) = \mathcal{H}(\boldsymbol{\theta}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda})$, thus the goal of HPO is to find $\boldsymbol{\lambda}^\star \triangleq \arg\min_{\boldsymbol{\lambda} \in \Omega} \mathcal{H}(\boldsymbol{\theta}_{\boldsymbol{\lambda}}, \boldsymbol{\lambda})$.

We specify these inner and outer objectives when treating real HPO tasks in Section 5.

3. `Hyperband` coupled with top-two algorithms

We propose a heuristic based on `Hyperband` and top-two variants of `ThompsonSampling` (`TTTS`, see Russo 2016) or `EI` algorithm (`TTEI`, see Qin et al. 2017). `Hyperband` is a bandit-based algorithm for HPO problem on top of a *fixed-budget best arm identification* (BAI) algorithm, namely `SequentialHalving` (Karnin et al., 2013). Similar to `Hyperband`, the heuristic runs several brackets of `TTTS` (or `TTEI`) with different number of configurations n and the same budget, we trade off between the number of configurations and the number of resources allocated to each configuration. Unlike `Hyperband` where the number of resources is fixed for each configuration in every bracket, here the number of resources is decided by the underlying subroutine.

The idea of using top-two algrithms is motivated by their being *anytime* for best arm identification. We first show some comparisons of different BAI algorithms, then we briefly introduce the top-two algorithms before presenting the heuristic in detail.

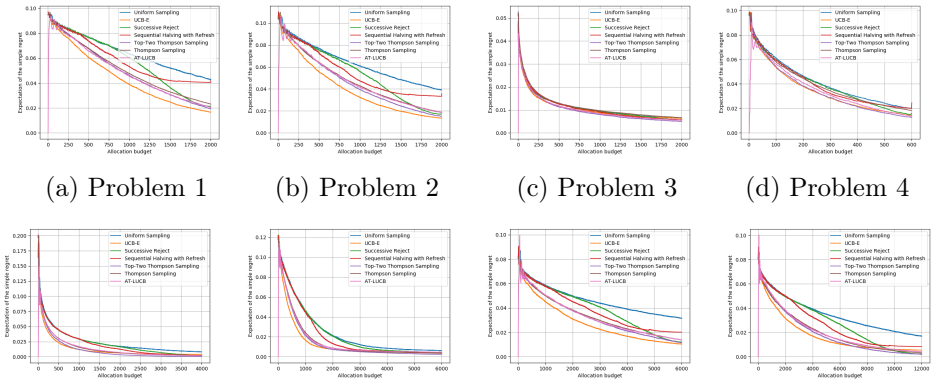
3.1 Best of best arm identification

We consider here some fixed-budget and anytime best arm identification algorithms, including `UniformAllocation` (Bubeck et al., 2009), `UCB-E` and `SuccessiveReject` (Audibert and Bubeck, 2010), fixed-budget version of `UGapE` (Gabillon et al., 2012), `SequentialHalving`, `ThompsonSampling` with a *most played arm* strategy of recommendation, `TTTS`, `AT-LUCB` (Jun and Nowak, 2016).

1. The loss function ℓ_2 can be different from the loss function ℓ_1 depending on the learning task.

We use 7 problem instances proposed by Audibert and Bubeck (2010), all settings consider only Bernoulli distributions, and we compare their trending *simple regret* averaged on 1000 trials. The results are shown in Fig. 1.

- Setting 1: $\mu_{\mathbf{1}} = 0.5, \mu_{\mathbf{2:20}} = 0.4, \text{budget} = 2000$
- Setting 2: $\mu_{\mathbf{1}} = 0.5, \mu_{\mathbf{2:6}} = 0.42, \mu_{\mathbf{7:20}} = 0.38, \text{budget} = 2000$
- Setting 3: $\mu = [0.5, 0.3631, 0.449347, 0.48125839], \text{budget} = 2000$
- Setting 4: $\mu = [0.5, 0.42, 0.4, 0.4, 0.35, 0.35], \text{budget} = 600$
- Setting 5: $\mu_{\mathbf{1}} = 0.5, \mu_{\mathbf{i}} = \mu_{\mathbf{1}} - 0.025i, \forall i \in \{2 \dots 15\}, \text{budget} = 4000$
- Setting 6: $\mu_{\mathbf{1}} = 0.5, \mu_{\mathbf{2}} = 0.48, \mu_{\mathbf{3:20}} = 0.37, \text{budget} = 6000$
- Setting 7: $\mu_{\mathbf{1}} = 0.5, \mu_{\mathbf{2:6}} = 0.45, \mu_{\mathbf{7:20}} = 0.43, \mu_{\mathbf{7:20}} = 0.38, \text{budget} = 6000$
- Setting 8: $\mu_{\mathbf{1}} = 0.5, \mu_{\mathbf{2:6}} = 0.45, \mu_{\mathbf{7:20}} = 0.43, \mu_{\mathbf{7:20}} = 0.38, \text{budget} = 12000$



(a) Problem 1 (b) Problem 2 (c) Problem 3 (d) Problem 4

(e) Problem 5 (f) Problem 6 (g) Problem 7 (h) Problem 8

Figure 1: Comparing different fixed-budget and anytime BAI algorithms.

In these experiments, `TTTS` are always beating or performing as well as its competitors.

3.2 Top-two algorithms

We briefly describe the top-two algorithms in the context of hyper-parameter tuning, where we consider each configuration of hyper-parameters as an arm. Top-two algorithms, as other Bayesian optimization approaches, make use of a prior distribution Π_1 over a set of parameters Φ, where for each $\boldsymbol{\phi} = \{\phi_{\boldsymbol{\lambda}} : \boldsymbol{\lambda} \in \Omega\} \in \Phi$, the hyper-parameter configuration $\boldsymbol{\lambda}$ is characterized by $\phi_{\boldsymbol{\lambda}}$. Based on a sequence of observations, we can update our beliefs to attain a posterior distribution Π_t. The posterior distribution can be computed by conjugacy when using common probability distributions for the prior. For example, `TTTS` uses correlated *one-dimensional exponential family* priors, and `TTEI` uses uncorrelated Gaussian priors.

At each time step t, top-two algorithms sample $\boldsymbol{\phi}$ from Π_t. With probability β, `TTTS` evaluates the arm $\boldsymbol{\lambda}_I$ such that $\phi_{\boldsymbol{\lambda}_I} = \max_{\boldsymbol{\lambda}\in\Omega} \widehat{\phi}_{\boldsymbol{\lambda}}$. Otherwise it samples a new $\boldsymbol{\phi}$ from Π_t until we obtain a different $J \neq I$ such that $\boldsymbol{\lambda}_J = \arg\max_{\boldsymbol{\lambda}\in\Omega} \widehat{\phi}_{\boldsymbol{\lambda}}$.

`TTEI` follows the same idea, but is slightly modified. With porbability β, `TTEI` evaluates the arm $\boldsymbol{\lambda}_I$ with the biggest *expected improvement* value (Mockus et al., 1978). Then with probability $1-\beta$, `TTEI` evaluates the arm $\boldsymbol{\lambda}_J$ with the biggest *relative* expected improvement value w.r.t. $\boldsymbol{\lambda}_I$.

3.3 Description of the heuristic

The heuristic we propose requires four inputs β, γ, B, and $s_{\max}$: (1) β is required by the underlying `TTTS` procedure, (2) γ characterizes how fast does the number of configurations we want to evaluate in each bracket shrink, (3) B is the total budget, and (4) $s_{\max}$ represents the index of the largest bracket (containing the largest number of configurations). Thus, in each bracket s, we dispose $n = \lceil \gamma^s (s_{\max}+1)/(s+1) \rceil$ configurations and an identical dedicated budget $T = \lfloor B/s_{\max} \rfloor$, and we run `TTTS` over these configurations. Each evaluation consists of training the classifier with one unit of resources w.r.t. a certain set of hyper-parameters.

Algorithm 1: Heuristic (based on `TTTS`)

Input : β; γ; B; $s_{\max}$
Initialize: $T = \lfloor B/s_{\max} \rfloor$; $L = \emptyset$; $t = 0$
1 **for** $s \leftarrow s_{\max}$ **to** 0 **do**
2 $n = \left\lceil \frac{s_{\max}+1}{s+1} \gamma^s \right\rceil$;
3 $\Omega =$ `get_configurations`(n);
4 $t = 0$;
5 // `TTTS`;
6 **while** $t < T$ **do**
7 `sample` $\widehat{\boldsymbol{\phi}} \sim \Pi_t$; $\boldsymbol{\lambda}_I \leftarrow \arg\max_{\boldsymbol{\lambda}\in\Omega} \widehat{\phi}_{\boldsymbol{\lambda}}$;
8 `sample` $b \sim$ Bernoulli(β);
9 **if** $b = 1$ **then**
10 $L = L \cup \{\mathcal{H}(\widehat{\boldsymbol{\theta}}_t, \boldsymbol{\lambda}_I)\}$;
11 **else**
12 `repeat sample` $\widehat{\boldsymbol{\phi}} \sim \Pi_t$; $\boldsymbol{\lambda}_J \leftarrow \arg\max_{\boldsymbol{\lambda}\in\Omega} \widehat{\phi}_{\boldsymbol{\lambda}}$ `until` $I \neq J$;
13 $L = L \cup \{\mathcal{H}(\widehat{\boldsymbol{\theta}}_t, \boldsymbol{\lambda}_J)\}$;
14 **end**
15 `update`(Π_t);
16 $t = t + 1$;
17 **end**
18 **end**
19 **return** *Configuration with the smallest intermediate loss seen so far*

The pseudo-code is shown in Algorithm 1. An important component that needs to be clarified is how to update the posterior belief (Line. 17 in Algorithm 1) in practice. We assume a Beta prior which is usually associated with Bernoulli bandits. Here in our case,

however, the reward (or more precisely, the loss) lies in $[0, 1]$, we thus need to adapt the `ThompsonSampling` process to the general stochastic bandits case. One way to tackle this is the binarization trick shown in Algorithm 2 inspired by Agrawal and Goyal (2011).

Algorithm 2: Detailed `TTTS` with Beta prior for general stochastic bandits

Input : $n \leftarrow |\Omega|$, α_0, β_0, β
Initialize: $\forall \boldsymbol{\lambda} \in \Omega, S_{\boldsymbol{\lambda}} = 0, F_{\boldsymbol{\lambda}} = 0$
for $t \leftarrow 1$ **to** T **do**
  $\forall \boldsymbol{\lambda} \in \Omega$, sample $\widehat{\phi}_{\boldsymbol{\lambda}} \sim \text{Beta}(S_{\boldsymbol{\lambda}} + \alpha_0, F_{\boldsymbol{\lambda}} + \beta_0)$; $\boldsymbol{\lambda}_I \leftarrow \arg\max_{\boldsymbol{\lambda} \in \Omega} \widehat{\phi}_{\boldsymbol{\lambda}}$;
  `sample` $b \sim \text{Bernoulli}(\beta)$;
  if $b = 1$ **then**
    `evaluate` $\boldsymbol{\lambda}_I \rightarrow \mathcal{H}(\widehat{\boldsymbol{\theta}}_t, \boldsymbol{\lambda}_I)$;
  else
    `repeat` $\forall \boldsymbol{\lambda} \in \Omega$, `sample` $\widehat{\theta}_c \sim \text{Beta}(S_{\boldsymbol{\lambda}} + \alpha_0, F_{\boldsymbol{\lambda}} + \beta_0)$; $\boldsymbol{\lambda}_J \leftarrow \arg\max_{\boldsymbol{\lambda} \in \Omega} \widehat{\phi}_{\boldsymbol{\lambda}}$ `until` $I \neq J$;
    `set` $I \leftarrow J$;
    `evaluate` $\lambda_I \rightarrow \mathcal{H}(\widehat{\boldsymbol{\theta}}_t, \boldsymbol{\lambda}_I)$;
  end
  `sample` $r_t \sim \text{Bernoulli}(1 - \mathcal{H}(\widehat{\boldsymbol{\theta}}_t, \boldsymbol{\lambda}_I))$;
  if $r_t = 1$ **then**
    $S_I \leftarrow S_I + 1$;
  else
    $F_I \leftarrow F_I + 1$;
  end
  $t = t + 1$;
end

4. Theorectical perspectives

4.1 Sample proportion convergence

5. Experimental results

We present some experiments for both non-stochastic and stochastic setting. For the non-stochastic setting, we assess different HPO algorithms by training classifiers for which the inner objective is evaluated via *stochastic gradient descent* (`SGD`) on the `MNIST` dataset[2]. For the stochastic setting, we consider different classical machine learning models (such as `SVM`, `KNN`, etc) that are trained on different datasets from `UCI` machine learning repository[3].

2. http://yann.lecun.com/exdb/mnist/
3. https://archive.ics.uci.edu/ml/datasets.html

5.1 Hyper-parameter tuning as NIAB problem

We first consider HPO as a NIAB problem, which means with each unit of resources, we don't need to train the classifier into completion, and each time a new unit of resources is allocated to the same hyper-parameter configuration, we just continue the unfinished training phase with this configuration.

In this part, we train logistic regression, multi-layer perceptron (`MLP`) and convolutional neural networks (`CNN`) on the `MNIST` dataset using mini-batch `SGD` as inner optimization method[4]. The type of resources considered is *epoch* that consists of one full training cycle on the whole training set. Note that this is similar to the `Hyperband` paper where one unit of resources corresponds to 100 mini-batch iterations for example. One epoch may contain a various number of mini-batch iterations depending on the mini-batch size.

The dataset is pre-split into three parts: training set $\mathcal{D}_{\text{train}}$, validation set $\mathcal{D}_{\text{valid}}$ and test set $\mathcal{D}_{\text{test}}$. Hyper-parameters to be taken into account for each classifier are listed below in Table 1. For logistic regression, the hyper-parameters to be considered are learning rate and mini-batch size (since we are doing mini-batch `SGD`). For `MLP`, we take into account an additional hyper-parameter which is the l_2 regularization factor. For `CNN`, we take into account the number of kernels used in the two convolutional-pooling layers.

Classifier	**Hyper-parameter**	**Type**	**Bounds**
`logistic regression`	learning_rate	$\mathbb{R}^+$	$[10^{-3}, 10^{-1}]$ (log-scaled)
	batch_size	$\mathbb{N}^+$	$[1, 1000]$
`MLP`	learning_rate	$\mathbb{R}^+$	$[10^{-3}, 10^{-1}]$ (log-scaled)
	batch_size	$\mathbb{N}^+$	$[1, 1000]$
	l_2_reg	$\mathbb{R}^+$	$[10^{-4}, 10^{-2}]$ (log-scaled)
`CNN`	learning_rate	$\mathbb{R}^+$	$[10^{-3}, 10^{-1}]$ (log-scaled)
	batch_size	$\mathbb{N}^+$	$[1, 1000]$
	k_2	$\mathbb{N}^+$	$[10, 60]$
	k_1	$\mathbb{N}^+$	$[5, k_2]$

Table 1: Hyper-parameters to be optimized.

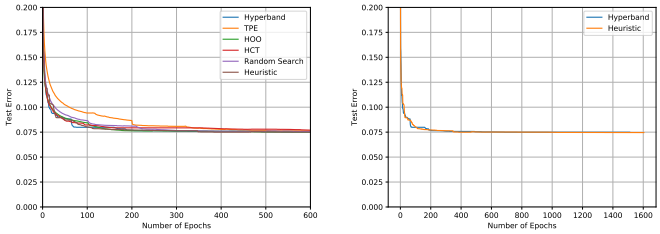
Inner and outer objectives In this part, we focus on neural network-typed classifiers, for which we want to maximize the likelihood of the training set $\mathcal{D}_{\text{train}}$ under a model parameterized by $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ with respect to a set of hyper-parameters $\boldsymbol{\lambda}$, where $\mathbf{W}$ corresponds to the weight matrix and $\mathbf{b}$ is the bias vector:

$$L_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) = \sum_{i=1}^{|D_{\text{train}}|} \log(\mathbb{P}(Y = y_i|\mathbf{x}_i, \boldsymbol{\theta}).$$

This leads to the following definition of inner objective:

$$\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \triangleq -L_{\boldsymbol{\lambda}}(\boldsymbol{\theta}).$$

4. This part of code (code for classifers with eventual usage of GPU) is based on code available at http://deeplearning.net/



(a) All methods (b) `Hyperband` and the heuristic

Figure 2: Comparing different hyper-parameter optimization algorithms on logistic regression, trained on `MNIST` dataset.

At each time step, we allocate one unit of resources (one epoch) to the classifier, which is trained on the training set $\mathcal{D}_{\text{train}}$. The trained model is then used to predict output values $\widehat{y}$ and $\widetilde{y}$ respectively over validation set $\mathcal{D}_{\text{valid}}$ and test set $\mathcal{D}_{\text{test}}$ for each data point in these two sets. We then define the outer objective as the number of misclassified data point by the model, a.k.a. the averaged zero-one loss on the validation set:

$$\mathcal{H}(\boldsymbol{\theta}, \boldsymbol{\lambda}) \triangleq \frac{1}{|\mathcal{D}_{\text{valid}}|} \sum_{i=1}^{|\mathcal{D}_{\text{valid}}|} \mathbb{1}_{\{\widehat{y}_i \neq y_i\}}.$$

Results During the experiment, we keep track of the best validation error and its associated test error $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i=1}^{|\mathcal{D}_{\text{test}}|} \mathbb{1}_{\{\widetilde{y}_i \neq y_i\}}$. At each time step, if the new validation error is smaller than the current best validation error, then we update the best validation error, and report the new test error. Otherwise we just report the test error associated with the previous best validation error. All plots here (from Fig. 2 to Fig. 3) are averaged on 10 trials of experiments.

Discussion In the current setting, `Hyperband` seems to be a plausible choice since it may explore more configurations compared to other algorithms. And comparing `Hyperband` and the heuristic, we can see that they almost performs as well as each other. There are two major observations here. First, the heuristic seems to be converging more smoothly than `Hyperband`, this is because `Hyperband` focuses on evaluating one of the configurations after its round-robin tour among all the configurations in the bracket, which leads to a straight drop of the output loss. The second observation is that the heuristic seems to have a slightly better output at the end. This could be due to the fact that `TTTS` is likely to evaluate the eventual best configuration more frequently than `SequentialHalving`.

5.2 Hyper-parameter tuning as SIAB problem

Now we consider the setting where each evaluation of a set of hyper-parameters consists of a complete training over the training set. We use adaptive boosting (`AdaBoost`), gradient boosting machine (`GBM`), k-nearest neighbors (`KNN`), `MLP`, support vector machine (`SVM`) from `scikit-learn`, and we evaluate them on several datasets from `UCI` dataset archive (e.g.

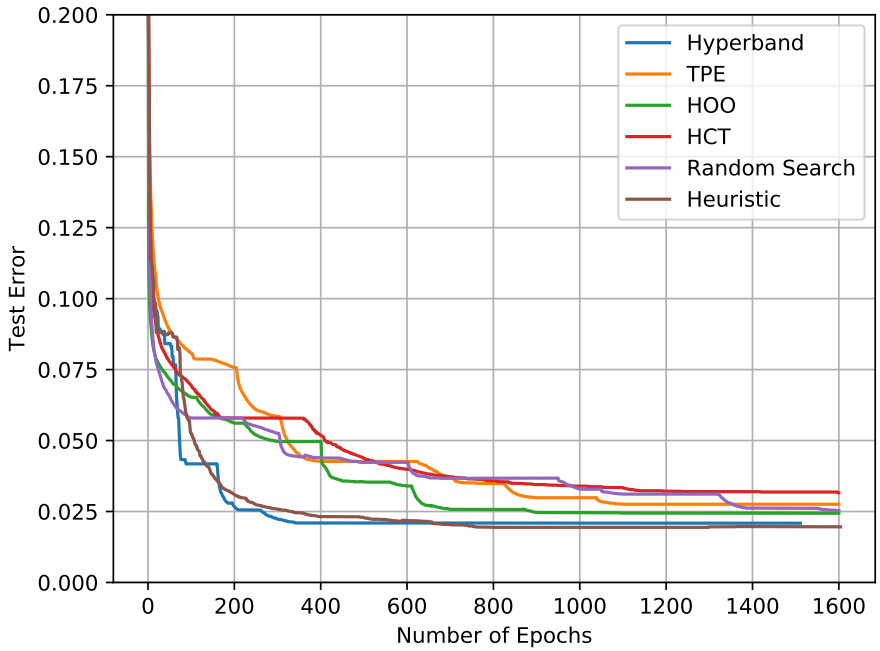


Figure 3: Comparing different hyper-parameter optimization algorithms on `MLP`, trained on `MNIST` dataset.

Wine, Breast Cancer, etc). They are all pre-split into a training set $\mathcal{D}_{\text{train}}$ and a test set $\mathcal{D}_{\text{test}}$. Hyper-parameters to be optimized are listed below in Table 2.

Classifier	**Hyper-parameter**	**Type**	**Bounds**
`AdaBoost`	`learning_rate`	$\mathbb{R}^+$	$[10^{-5}, 10^{-1}]$
	`n_estimators`	Integer	$\{5, \ldots, 200\}$
`GBM`	`learning_rate`	$\mathbb{R}^+$	$[10^{-5}, 10^{-2}]$
	`n_estimators`	Integer	$\{10, \ldots, 100\}$
	`max_depth`	Integer	$\{2, \ldots, 100\}$
	`min_samples_split`	Integer	$\{2, \ldots, 100\}$
`KNN`	k	Integer	$\{10, \ldots, 50\}$
`MLP`	`hidden_layer_size`	Integer	$[5, 50]$
	`alpha`	$\mathbb{R}^+$	$[0, 0.9]$
`SVM`	C	$\mathbb{R}^+$	$[10^{-5}, 10^{5}]$ (log-scaled)
	γ	$\mathbb{R}^+$	$[10^{-5}, 10^{5}]$ (log-scaled)

Table 2: Hyper-parameters to be optimized.

In this part, we define different inner objective for classification and regression problems. We use the logarithmic loss, also known as cross-entropy for classification problems:

$$\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \triangleq -\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i=1}^{|\mathcal{D}_{\text{train}}|} \sum_{j=1}^{m} y_i^{(j)} \log(\widehat{p}_i^{(j)}),$$

where $\widehat{p}_i^{(j)}$ is the predicted probability of a sample i belonging to class j, and m is the number of classes considered. For regression problems, we use the typical mean squared error:

$$\mathcal{L}_{\boldsymbol{\lambda}}(\boldsymbol{\theta}) \triangleq -\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{i=1}^{|\mathcal{D}_{\text{train}}|} (y_i - \widehat{y}_i)^2 .$$

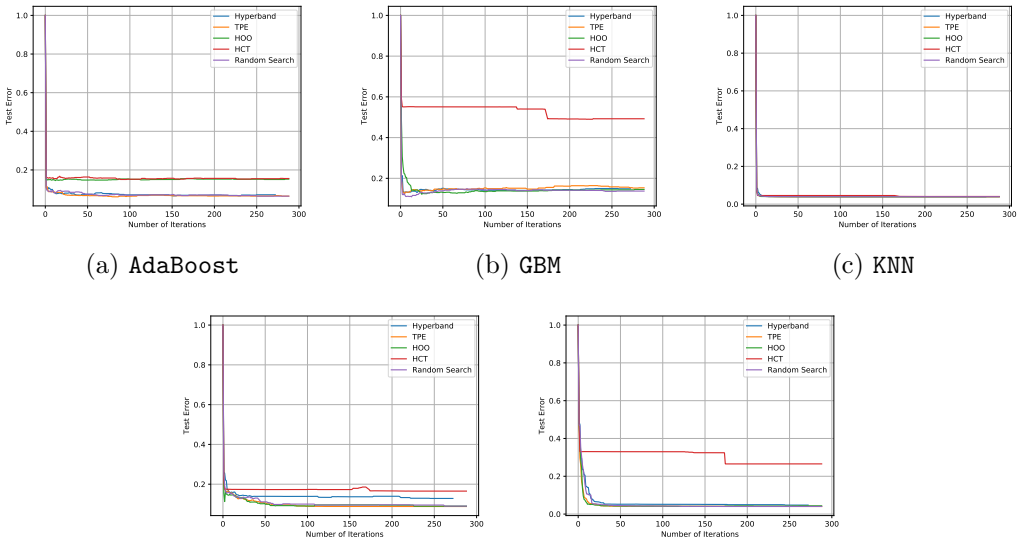
Regarding the outer objective, we choose to perform a shuffled $k = 5$ cross-validation scheme on $\mathcal{D}_{\text{train}}$ at each time step. In practice, this means that we fit 5 models with the same architecture to different train/validation splits and average the loss results in each. More precisely, for every cross-validation split $\texttt{cv}_j, j = 1 \dots 5$, we get a loss $\frac{1}{n} \sum_{i=1}^{n} \left(y_i^{(j)} - \widehat{y}_i^{(j)} \right)^2$, where n is the number of data points in the validation set (here we take MSE as an example, it's the same for log-loss). The outer objective is thus

$$\mathcal{H}(\boldsymbol{\lambda}, \boldsymbol{\theta}) \triangleq \frac{1}{5n} \sum_{j=1}^{5} \sum_{i=1}^{n} \left(y_i^{(j)} - \widehat{y}_i^{(j)} \right)^2 .$$

Note that under the current setting, `Hyperband` needs to be adapted. Assuming that each complete training of a classifier w.r.t. some hyper-parameter configuration is an i.i.d. sample, the elimination phase of the underlying `SequentialHalving` is carried out according to the averaged loss of previous samplings.

Results Like in the previous part, we also report the test error $\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{i=1}^{|\mathcal{D}_{\text{test}}|} \left(y^{(i)} - \widetilde{y}_{\text{pred},t}^{(i)} \right)^2$ on the holdout test set $\mathcal{D}_{\text{test}}$ here.

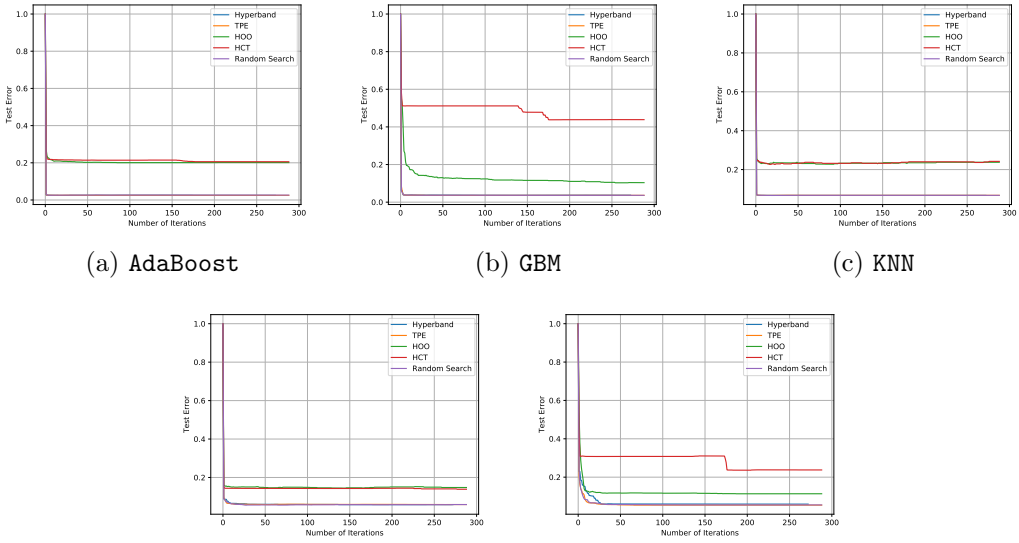
Under this experimental environment, "keep training" does not make sense anymore. Thus for `HOO`, `TPE` and `RandomSearch`, we only need to evaluate each configuration once, contrarily to what we did in the previous setting. While for `Hyperband`, we still need to evaluate each configuration for a certain times based on R and $s_{\max}$. Each plot here is averaged on 20 runs of experiments. Fig. 4 displays the results for the Wine dataset, and Fig. 5 displays the results for the Brest Cancer dataset.



(a) `AdaBoost` (b) `GBM` (c) `KNN`

(d) `MLP` (e) `SVM`

Figure 4: Comparing different hyper-parameter optimization algorithms on different classifiers, trained on Wine dataset.



(a) `AdaBoost` (b) `GBM` (c) `KNN`

(d) `MLP` (e) `SVM`

Figure 5: Comparing different hyper-parameter optimization algorithms on different classifiers, trained on Breast Cancer dataset.

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, 2011.

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

Mohammad Gheshlaghi Azar, Alessandro Lazaric, and Emma Brunskill. Online stochastic optimization under correlated bandit feedback. In *Proceedings of the 31st International Conference on Machine Learning (ICML)*, pages 1557–1565, 2014.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

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.

Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. 2010.

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.

Chih-Chun Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):1–27, 2011.

Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

Luca Franceschi, Paolo Frasconi, Saverio Salzo, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. 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.

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.

Kwang-Sung Jun and Robert Nowak. Anytime exploration for multi-armed bandits using confidence information. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48, pages 974–982, 2016.

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.

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, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. Technical report, 2016.

Jonas Mockus, Vytautas Tiešis, and Antanas Žilinskas. The application of Bayesian methods for seeking the extremum. *Towards Global Optimisation 2*, pages 117–129, 1978.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Chao Qin, Diego Klabjan, and Daniel Russo. Improving the expected improvement algorithm. In *Advances in Neural Information Processing Systems 30 (NIPS)*, pages 5381–5391, 2017.

Daniel Russo. Simple Bayesian algorithms for best arm identification. In *Proceeding 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.

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.

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa A. Patwary, Prabhat, and Ryan P. Adams. Scalable Bayesian optimization using deep neural networks. In *Proceedings of the 32nd International conference on Machine Learning (ICML)*, 2015.

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.