Advanced BO methods for Automatic Hyperparameter Tuning

Kuluhan Binici
Shivam Aggarwal

1 Introduction

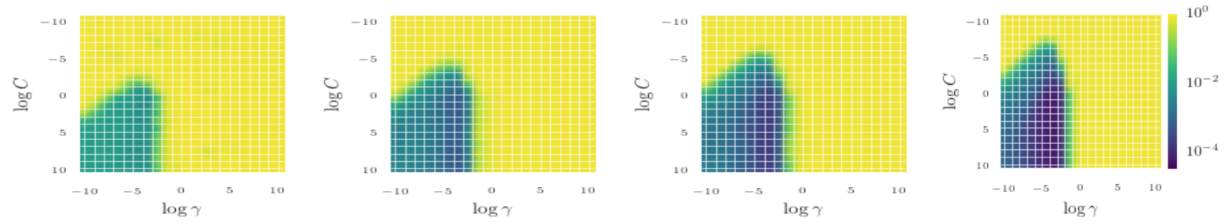
The majority of machine learning techniques contain specific parameters that can not be optimized during the learning process. These are called hyper-parameters, and searching optimal configurations for them is of significance for maximizing the performance on target learning tasks. Some traditional approaches are based on exhaustively evaluating configurations in a specified search range based on the test loss they yield after being used to train models. Depending on the machine learning model and dataset of interest, these evaluations can be highly costly in terms of wallclock time. For instance, finishing 90-epoch ImageNet training with ResNet-50 neural network model on an NVIDIA M40 GPU takes 14 days. Therefore, searching for more efficient and principled hyper-parameter optimization (HPO) methods has become a significant research direction. Recently, Bayesian Optimization techniques are being employed to carry out this task. Due to their ability to learn from a small amount of data, Gaussian Process optimization techniques are proper candidates. Moreover, their properties allow evaluation points to be acquired in a careful ordering that can substantially reduce the search time to converge to optimal configurations. In this report, we will discuss two recent Bayesian optimization techniques that accelerate HPO for machine learning models. These works are Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets (FABOLAS) [4] and Robust and Efficient Hyperparameter Optimization at Scale (BOHB) [2].

2 FABOLAS

In the following sections we will discuss the motivation behind FABOLAS, introduce the algorithm and share experimental evaluations to study its effectiveness.

2.1 Motivation

The proposed method stems from a hypothesis that hyper-parameter configurations can be evaluated without having to train a neural network model on the entire target dataset. Instead, fractions of the entire collection of data samples can be used. First, the authors validate this hypothesis via a toy experiment. In the experiment, the optimal hyper-parameters for an SVM are searched based on the evaluations made by using different fractions of the MNIST dataset. Various subsets with relative sizes $s \in \{1/512, 1/256, 1/128, ..., 1/4, 1/2, 1\}$ are used to evaluate 400 configurations of the *kernel*(γ) and *regularization*(C) hyper-parameters. The outcome of this experiment appears in Figure 1. In the figure, configurations are marked with colors based on the test losses they yield on a logarithmic scale.



(a) $s = 1/128$ (b) $s = 1/16$ (c) $s = 1/4$ (d) $s = 1$

Figure 1: Toy experiment to compare hyper-parameter configuration evaluations done by using various fractions of MNIST dataset.

From the figure, it can be observed that the ordering among test losses obtained by different configurations is consistent when the evaluations are done via various fractions of the target dataset. After validating their hypothesis with this experiment, the authors propose incorporating the choice of evaluation data subset size in the hyper-parameter search process. This way, configurations can be evaluated on small but sufficient fractions of the dataset, which would be much faster and computationally cheaper than full-scale evaluations.

2.2 Proposed GP Model

To explicitly optimize the computational cost of evaluating a configuration, the authors choose to model it as a gaussian $f_c(x, s) \sim \mathcal{N}(0, K_c)$, like they do for testing loss $f_f(x, s) \sim \mathcal{N}(0, K_f)$. As in any GP modelling task, a proper covariance function should be defined to optimally represent the similarity among observations. The authors preferred to use the function from Equation 1 for this purpose.

$$k((x, s), (x', s')) = k_{5/2}(x, x') \cdot (\phi^T(s) \cdot \Sigma_\phi \cdot \phi(s')) \tag{1}$$

In Equation 1, Matérn 5/2 ($k_{5/2}$) kernel is used to measure the similarity between two configurations (i.e. x and x'). Due to being twice differentiable and allowing the costly calculation of Bessel function to be evaded, Matérn 5/2 is commonly used in GP optimization tasks. This kernel is multiplied with the covariance function of subset sizes to correlate the two pairs ((x, s) with (x', s')). While the covariance function is described as Equation 1 in the paper, the official implementation contains a slightly different version for the $(\phi^T(s) \cdot \Sigma_\phi \cdot \phi(s'))$ part (see Equation 2).

$$k((x, s), (x', s')) = k_{5/2}(x, x') \cdot (b\phi^T(s) \cdot \phi(s') + a) \tag{2}$$

In Equation 2, a and b are tunable parameters that are optimized in conjunction with neural network training via gradient-based updates.

The selection of baseline function ϕ differs for GP models of test loss and computational cost. This is to better incorporate the domain knowledge related to the impact of subset size over these variables, respectively. Intuitively, as the number of samples used to train a neural network model increases, the test loss is expected to decrease. Therefore, ϕ_c is selected as $(1, s)^T$. As for computational cost, it is expected to increase together with the number of training samples. Therefore, ϕ_f is selected as $(1, (1 - s)^2)^T$.

Here we also note that using a simple dot product to represent the joint correlation of x and s in

Equation 1 is questionable. This is because the same recipe is used to represent covariances of both test loss and computational cost which imposes the unlikely assumption that changes in covariance related to configurations (x) would have the same impact on test loss and the computational cost.

2.3 Acquisition Function

2.3.1 Preliminaries

In this section we will introduce two strategies for selecting evaluation points that inspired the acquisition function used in FABOLAS.

Entropy Search (ES) The goal in ES is to find evaluation points that maximize the information about the optimum of the objective function. If the target is to minimize the objective function, the optimum is modelled with the probability distribution $p_{min}(x|D) = p(x \in argmin_{x' \in X} f(x')|D)$. Moreover, the acquisition function can be denoted as in Equation 3.

$$a_{ES}(x) = \mathbb{E}_{p(y|x,D)} \left[\int p_{min}(x'|D \cup \{(x,y)\}) \cdot log \frac{p_{min}(x'|D \cup \{(x,y)\})}{u(x')} dx' \right] \tag{3}$$

The integral term corresponds to the Kullback-Leibler (KL) divergence between the p_{min} and uniform distributions. From a high level, this can be interpreted as maximizing the reduction in entropy of p_{min} since the uniform distribution has the maximum entropy. Such reduction in entropy is also referred to as information gain.

Multi-Task Bayesian Optimization (MTBO) When a Bayesian optimization framework requires evaluating data points on multiple correlated tasks, it is referred to as MTBO. The time it takes for each evaluation differs based on the task, which is unknown prior to the evaluation. Therefore, the computational cost is also modeled as a gaussian and included in the optimization. The acquisition function used for MTBO is almost identical to ES with the addition of the cost term (see Equation 4) to motivate the acquisition of data point-task pairs that are computationally cheaper to evaluate. In [6] the tasks were represented as discrete variables.

$$a_{MT}(x,t) = \frac{1}{c(x,t)} \mathbb{E}_{p(y|x,t,D)} \left[\int p^{t_*}_{min}(x'|D \cup \{(x,t,y)\}) \cdot log \frac{p^{t_*}_{min}(x'|D \cup \{(x,t,y)\})}{u(x')} dx' \right] \tag{4}$$

2.3.2 FABOLAS acquisition function

Evaluating configurations on different subsets of the training data can be viewed as an MTBO problem. With this observation, the authors build on top of the MTBO acquisition function while coming up with their own for FABOLAS. They let a significant part of the function remain the same, with the only additional term of the computational cost of updating the GP model. This way, while deciding on the following evaluation points, those that take the least amount of total wall-clock time are preferred. The final version of the acquisition function employed in the paper appears below in Equation 5.

$$a_F(x,s) = \frac{1}{c(x,s) + c_{overhead}} \mathbb{E}_{p(y|x,s,D)} \left[\int p^{s=1}_{min}(x'|D \cup \{(x,s,y)\}) \cdot log \frac{p^{s=1}_{min}(x'|D \cup \{(x,s,y)\})}{u(x')} dx' \right] \tag{5}$$

Different than MTBO, in FABOLAS the tasks (subset sizes) are defined as continuous random variables and can take any real value in the specified interval.

2.4 Algorithm

The overall optimization procedure is as described below in Algorithm 1

Algorithm 1 FABOLAS

Initialize data D_0 using an initial design.
for t=1,2,... **do**
  Fit GP models for $f(x, s)$ and $c(x, s)$ on data D_{t-1}.
  Choose (x, s) by maximizing the acquisition function in Equation 5.
  Evaluate $y_t \sim f(x_t, s_t) + \mathcal{N}(0, \sigma^2)$, also measuring cost $z_t \sim c(x_t, s_t) + \mathcal{N}(0, \sigma^2)$.
  Augment the data $D_t = D_{t-1} \cup \{(x_t, s_t, y_t, z_t)\}$.
  Choose incumbent $\hat{x_t}$ based on the predicted loss at $s = 1$ of all $\{x_1, x_2, ..., x_t\}$.
end for

The process begins by sampling a group of initial evaluation points and fitting the GP models on them. Later at each iteration a new configuration-subset size pair is sampled based on the acquisition function from Equation 5. The machine learning model is trained on a subset of the training set of specified size and according to the specified hyper-parameter configuration. The performance over the test set and training time are recorded to augment the GP observations. This process repeats until the pre-determined number of steps is reached.

2.5 Experimental Evaluation

In this section we discuss the experimental evaluation given in the original paper. Moreover, we also share some additional experiments that we conducted by re-implementing the algorithm.

2.5.1 Baselines

The baseline methods that the authors provided comparison with are ES, MTBO, Expected Improvement (EI), random search and Hyperband. We already introduced the first two in Section 2.3 and will give a detailed description of Hyperband later, in Section 4.1.2.

Expected Improvement (EI) The main goal of EI strategy is to acquire evaluation points that are expected to improve the already-observed maxima/minima of the objective function, the most. For a minimization task, this goal can be expressed as,

$$a_{EI}(x|D_n) = \mathbb{E}_p \left[max(f_{min} - f(x), 0) \right] \tag{6}$$

where f_{min} represents the previously observed minimum of the objective function.

Random Search As the name implies, random search targets to converge to the optimal configuration by drawing evaluation points randomly from the specified search space.

2.5.2 Datasets

In the original paper, 5 image classification datasets were used for benchmarks. These are MNIST, SVHN, CIFAR10, covers types and vehicles. **MNIST** contains 70,000 32x32 grayscale images from ten hand-written digit classes. **SVHN** is a colored digit classification dataset consisting of over 600,000 labeled images. **CIFAR10** contains RGB images from ten classes (3x32x32). **Cover types** contains 581,012 RGB samples from 54 forest cover type classes. **Vehicles** dataset consists of more than 50,000 images of vehicles.

To extend the experimental evaluation, we tested FABOLAS on **Fashion MNIST** dataset which is an image classification dataset of 70,000 grey-scale cloth images from 10 different categories.

2.5.3 Support Vector Machine (SVM) Experiments

For the first set of experiments, the authors compare their method with all baselines on the task of hyper-parameter optimization for an SVM. The discretized configuration grid from the toy experiment in Section 2.1 is used as the search space. The results of these experiments are given in Figure 2.

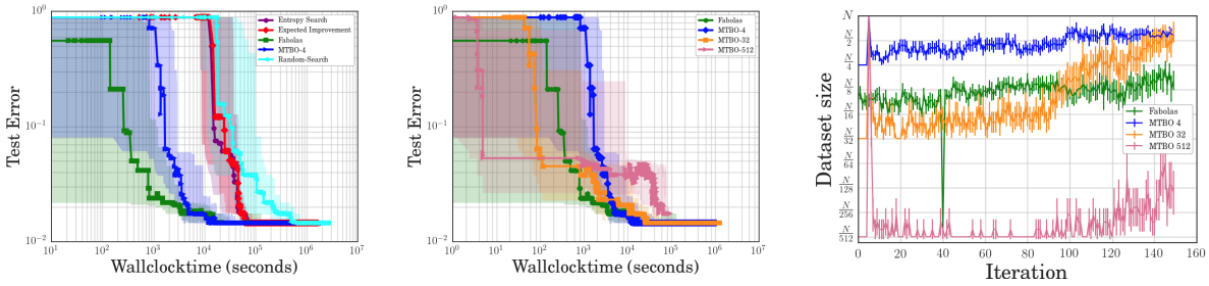


Figure 2: Performance comparison of various methods on hyper-parameter search for an SVM trained MNIST dataset. MTBO-X means that the minimum fraction of the dataset that can be used for evaluation is 1/X.

The first two plots (from the left) in Figure 2, demonstrate that FABOLAS can converge to the optimal configuration faster than other baselines. It is counter-intuitive that the MTBO-512 performs inferior to other alternatives with smaller search space in the middle plot. The authors attribute this to the much larger amount of evaluations needed to be made which makes it more challenging to locate the optimum.

For the next set of experiments, the constraint of limiting the search space to a discretized grid is removed. The comparisons given in Figure 3 are obtained when a continuous search space was used.

From the figure it can be observed that FABOLAS consistently outperforms the baselines across different datasets.

2.5.4 ConvNet Experiments

Further performance comparisons on finding optimal hyper-parameters for a 3-layer CNN and a ResNet are also given in the paper. The ordering among performances of different methods are the same as the SVM experiments and therefore we do not share the related figures here. Instead, we refer the reader to the original paper.

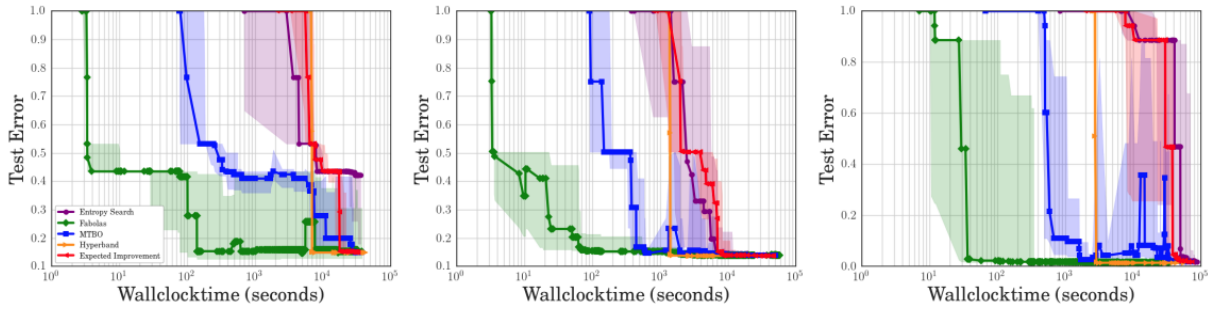


Figure 3: Performance comparison of various methods on hyper-parameter search for an SVM trained on 3 different datasets. The datasets used from left to right are vehicle, cover type and MNIST respectively.

3 Extended Experiments and Further Results

We extend the FABOLAS experiments to include HPO for a LeNet5 neural network model being trained on Fashion-MNIST [7] dataset.

3.1 Baselines

We selected grid and random search strategies as baselines to compare with FABOLAS. Grid search can be described as partitioning the search space into discrete points with equal intervals in between each point and iteratively evaluating all candidate points on the target task.

3.2 Implementation details

We implemented the FABOLAS GP model, main optimization script and neural network training objective function ourselves with the help of common libraries such as GPy, PyTorch, Numpy and etc. For the auxiliary operations such as defining the search space, acquisition function and optimizing GP parameters by gradients, we made use of the Emukit framework. We defined the optimization objective as finding the optimal learning rate for the stochastic gradient descent (SGD) optimizer that is used to train the LeNet model. The rest of the SGD hyper-parameters such as momentum and weight decay were fixed to 9×10^{-1} and 5×10^{-4} respectively. We limited the search space to subset sizes ranging from 2000 samples to 60,000 that correspond to the interval $s \in [1/30, 1]$. As for the range that we allowed the methods to search for learning rate was $[10^{-4}, 2 \times 10^{-1}]$. We set the total number of configurations to be evaluated as 100.

3.3 Instructions to run experiments

The code that we used in our experiments is shared together with this report. To run the experiments, clone the emukit framework from its GitHub repository via this link: https://github.com/EmuKit/emukit.git. Later extract all the files we provide in the folder 'FABOLAS_code' to the main folder of emukit. Then customize the search space, number of evaluation points and etc. based on preference by editing fabolas.py. Lastly run fabolas.py to reproduce experiments. Note, the emukit framework can also be installed by pip, however, when we tried that, we received some errors.

3.4 HPO performance comparison on Fashion MNIST

We acquired objective function values by training the LeNet on FashionMNIST subsets of specified size for a single epoch. The progression of maximum accuracy is plotted against wallclock time for all methods in Figures 4 and 5.

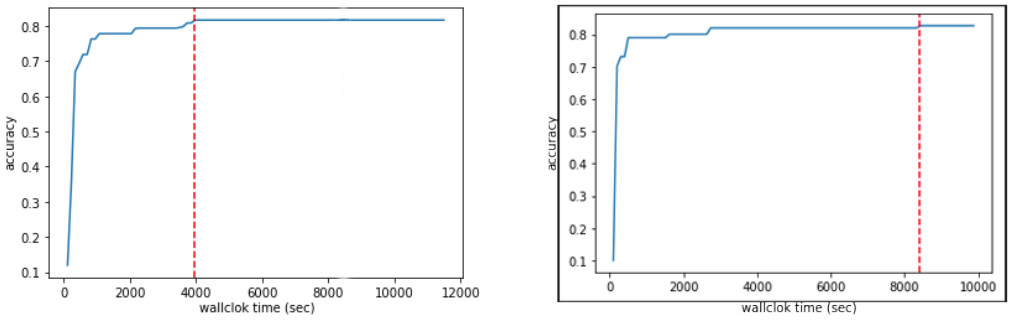


Figure 4: Achieved maximum accuracy over wallclock time for grid (left) and random search (right) methods.

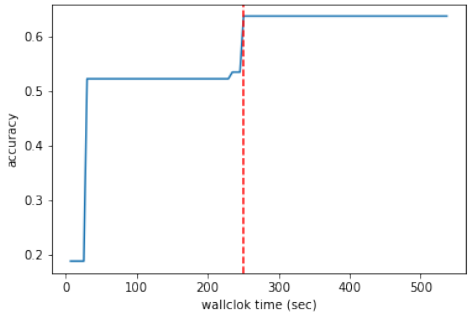


Figure 5: Achieved maximum accuracy over wallclock time for FABOLAS method.

From the plots, it can be observed that FABOLAS reaches optimal configuration in around 250 seconds which is much faster than the Grid and Random search as they take about 4000 and 8300 seconds to converge. Here we note that the maximum accuracy reached by FABOLAS being lower than the other methods is due to the neural network being trained only for a single epoch. While the number of training iterations is already tiny to reach complete accuracy, FABOLAS also uses only fractional subsets of the target dataset for training. Therefore, the maximum accuracy value it achieves during the search is expected to be lower than the baselines that perform full-scale evaluations on the entire dataset. The grid search performed better when compared to random search. This is due to the stochasticity of the random search in evaluating configurations. Since the convergence to the optimum is based on probability, it can take an arbitrarily long amount of time.

When we trained LeNet5 models for 10 epochs over the entire FashionMNIST training set with the optimal configurations we obtained by the three methods, the test accuracies achieved were 88.8%, 88.5% and 88.7% for FABOLAS, random search and grid search respectively.

Moreover, we also recorded the subset sizes that FABOLAS acquisition function evaluated which can be viewed in Figure 6.

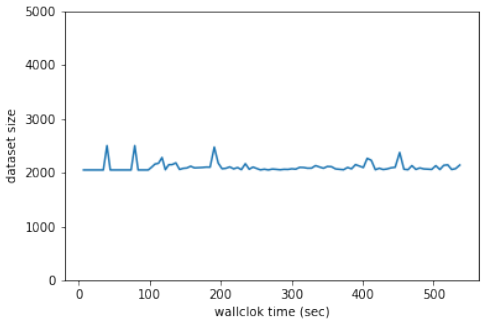


Figure 6: Subset sizes obtained by FABOLAS acquisition function for evaluation.

From the above figure it can be seen that the acquisition function preferred most evaluation points to have a subset size of around 2300 samples. This explains the speedup achieved against the baseline methods since they perform all the evaluations by using the entire training set of 60,000 samples.

In summary, these extended experiments are coherent with the results reported in the original paper and further validate the effectiveness of FABOLAS in accelerating hyper-parameter search.

4 Description of BOHB

This section introduces the BOHB algorithm and explains how BOHB combines Bayesian Optimization and Hyperband to get the best out of two worlds. As shown in Figure 7, the method outperforms both vanilla Bayesian Optimization and Hyperband with substantial speedup and most optimal model performance.

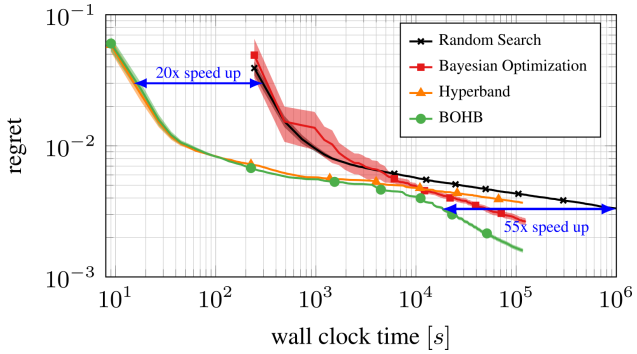


Figure 7: Performance of BOHB against Hyperband, Vanilla BO and Random Search [2]

For this purpose, the paper lays down several desirable traits for a practical HPO solution.

- **Strong Anytime Performance**: Most researchers only have a finite set of resources available at their disposal. Unless one is working in the industry, one would expect to achieve good model performance within small budgets (time, GPU resources, etc.).

- **Strong Final Performance**: It is apparent that for use in practice, the final performance of the model matters the most. Any HPO method ideally strives to achieve the same.
- **Scalability**: For realistic, workable solutions, HPO methods should be able to handle many dozens of hyperparameters and not just a few. Modern neural networks, for instance, have a variety of hyperparameters that need to be tuned.
- **Robustness & Flexibility**: Similarly, any HPO algorithm should be easily adaptable to different machine learning scenarios such as deep reinforcement learning and SVM.
- **Simplicity**: Simplicity, as the paper defines, is a virtue. It is an important trait for the democratization of the HPO methods to a general user.
- **Effective Use of Parallel Resources**: With big CPU clusters, HPO methods have a lot of parallel resources available for utilization and should be able to parallelize the method across different worker threads.
- **Computational efficiency**: Evaluating model accuracy and loss value for a given set of hyperparameters is a computationally expensive process. The cubic complexity of GP-based HPO methods further aggravates the cost of function evaluations. Hence, HPO algorithms should be critical of this component.

In the next section, we will first discuss two multi-fidelity optimization algorithms: successive halving and Hyperband, elaborate on their pros and cons and finally outline how BOHB leverages these concepts concurrently.

4.1 Preliminaries

4.1.1 Successive Halving

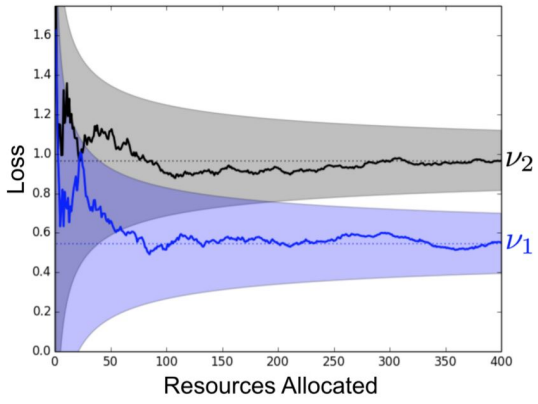


Figure 8: Validation loss across two different evaluations over time [5]

The underlying idea behind multi-fidelity optimization algorithms is to exploit cheaper fidelities of the objective function. Successive halving (SH) [3] is one such method and assumes that all configurations could

be stopped early to obtain their validation score. Precisely, SH consists of n different configurations sampled randomly and a total budget B. Here, budget usually refers to the total number of iterations or training epochs or the fraction of the validation dataset. The method keeps the best half of the configurations in each iteration and discards the other half of the not-good-enough configurations. It repeats these steps until only one single configuration remains. This method is generally fast; however, it lacks robustness. It is not very straightforward for a fixed budget B to decide whether one should consider many configurations with a small average training time or consider a small number of configurations with longer training cycles.

For instance, consider Figure 8. In this experiment, two sets of hyperparameter configurations are evaluated over time. Now, it is impossible to differentiate between the two sets of configurations until the point of divergence. Clearly, if SH stops function evaluation early enough, one would choose v_1. However, with more resources available, v_2 tends to outperform v_1 by a large margin. This trade-off between the number of configurations and the number of cuts becomes a major limitation in SH.

4.1.2 Hyperband

Hyperband (HB) [5] overcomes the limitations posed by Successive Halving based on a very simple idea. Building on top of SH, HB considers several possible values of n for a fixed budget B. HB repeatedly calls Successive Halving to discover the best out of n randomly-sampled configurations. In essence, HB performs a grid search over a feasible value of n.

The two inputs: (i) R, indicating the maximum amount of resources that can be assigned to any configuration, and (ii) η, controlling the proportion of configurations to be discarded after in each round of SH together determine the number of brackets to be considered. Here, brackets indicate the total number of different SH runs. Concretely, if $s_{max} = \lfloor \log_\eta R \rfloor$, then a total of $s_{max} + 1$ different brackets will be run.

	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

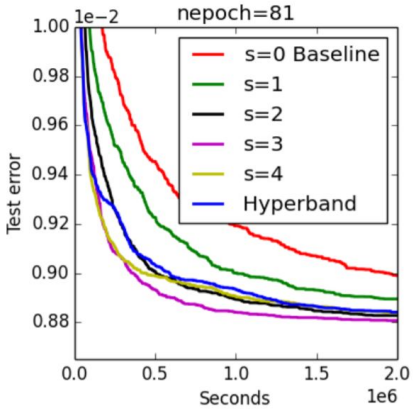


Figure 9: Left: Different brackets of HYPERBAND when $R = 81$ and $\eta = 3$ and Right: Performance of individual brackets [5]

For an example, refer to Figure 9. Here, $R = 81$, and η is set as 3, resulting in $s_{max} = 4$. This results in a total of 5 brackets of SH. For each s, the algorithm runs one separate SH run. Moreover, each SH run has the same total budget. Hence, they all take the same execution time. For $s = 4$, the algorithm stops the first 81 only after one epoch, repeats the steps, and terminates with one final configuration running for 81 epochs. The algorithm balances the trade-off between the number of different runs and the number of

cuts by sweeping through these values of s. As shown in Figure 9 on the right, performance comparison of Hyperband against individual SH runs. HB finds a sweet spot between very aggressive evaluations with many configurations on the smallest budget and a small number of configurations on the total budget. Neither the individual SH run with $s = 4$ nor the one with $s = 0$ gives the most optimal results. The best results are obtained when $s = 3$, something not very easy to find.

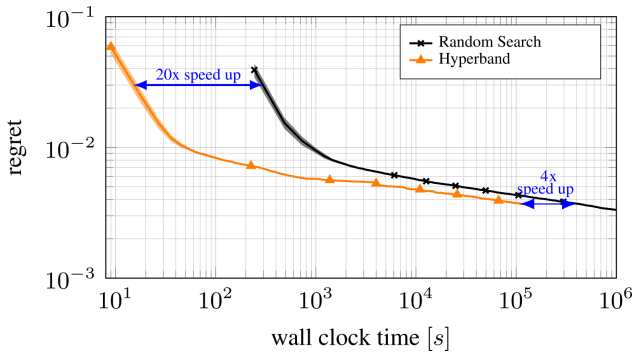


Figure 10: Performance of Hyperband over Random Search [5]

Overall, HB performs very well in practice. Notably, for small to medium budgets, HB outperforms random search and even vanilla BO methods. In terms of the desirable traits for a practical BO method, the technique offers flexibility, computational efficiency, scalability, and impressive parallelization ability. Most importantly, the technique provides strong anytime performance as shown in Figure 10. The method provides an ample opportunity to abruptly stall the hyperparameter search process at any time and result in good enough configurations.

However, the method relies on randomly-drawn configurations at the beginning of each SH run. For this reason, with large enough budgets, its advantage over random search diminishes very quickly. Hence, the method lacks a "strong anytime performance" trait. Unlike model-based approaches such as BO, Hyperband does not learn from its past results. It does not incorporate the knowledge learned so far to improve the configuration selection process for the next set of SH runs.

BOHB explores these limitations in HB and proposes a model-based approach using Bayesian Optimization to select configurations at the start of each HB iteration. BOHB relies on Tree-structured Parzen Estimators (TPE) to model the configuration selection framework, which we will discuss next.

4.1.3 BO using Tree Parzen Estimator (TPE)

Tree-structured Parzen Estimator (TPE) [1] is a non-conventional Bayesian Optimization method. Intuitively, "tree-structured" comes from the fact that the hyperparameter configuration space can be visualized as a tree. In such a tree-like space, nodes are potentially modeled as different hyperparameters, and a set of nodes collectively define child nodes. For instance, for a neural network, leaf variables such as the number of neurons in any particular layer of a network rely on node variables such as the total number of layers in the network. The general Gaussian-process-based methods model $p(y|x)$ directly, i.e., the probability of the model accuracy or loss value is given a set of hyperparameters. However, TPE takes an alternative approach by building $p(x|y)$ and $p(y)$. TPE leverages Kernel Density Estimators (KDE) to replace the distributions

of the configuration prior with non-parametric densities to model $p(x|y)$.

In general, density estimator methods seek to reconstruct the probability density function using a given set of data points. The histogram is a very simple and popular example of a density estimator algorithm. However, histogram generally results in discrete probability densities, which are not very useful in practice. Instead, kernel density estimator, one of the most famous density estimation methods, results in smooth, continuous probability densities. Consequently, KDE results in the following function:

$$p_n(x) = \frac{1}{nh} \sum_{i=1}^{n} K(\frac{X_i - x}{h}) \tag{7}$$

As can be seen in Equation 7, KDE generally has the following parameters: kernel function $K(x)$, specifying the shape of the distribution placed at each point and the kernel bandwidth h controlling the size of the kernel at each point and in turn responsible for the smoothness of the resultant curve. Note that the kernel function here is different from the standard kernel function in the Gaussian Processes. On a very high level, the idea is to convert each data point X_i into small bumps and then add them together to obtain the final density estimate.

Based on the above concept, TPE p(x|y) models two densities to separate the values of the hyperparameters based on a threshold. Consider a set of observations: $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(k)}, y^{(k)})\}$. TPE defines $p(y|x)$ using the following two density functions:

$$p(x|y) = \begin{cases} l(x) & \text{if } y^* > y \\ g(x) & \text{if } y^* \leq y \end{cases} \tag{8}$$

where $l(x)$ is the probability density function formed by the observations ($x^{(i)}$ such that the corresponding loss value $y < y^*$ and $g(x)$ is estimated using the other remaining observations. The core idea behind these two density functions is to characterize the set of hyperparameter configurations as "good" and "bad." The quantile value γ is usually used for this purpose such that $p(y^* > y) = \gamma$. While the GP-based Bayesian optimization follows an aggressive strategy (such as expected improvement) to model y^*, this method relaxes the condition based on a simple threshold value. TPE splits the model into two sets of observations to construct the models as outlined in Equation 8. This is usually achieved by simply sorting the observations based on their loss values or other considered parameters. Finally, to select the best candidate, i.e., the set of hyperparameters, TPEs maximize the ratio of $l(x)/g(x)$. As shown in the paper [1], the Expected Improvement is actually proportional to this ratio. Hence, maximizing the ratio gives the most optimal set of data points.

Key advantage of TPEs over GP-based BO methods is their flexibility in modeling both discrete and mixed continuous spaces with great ease. Due to the nature of the kernel, it is very convenient to model hyperparameter configurations. Moreover, the model construction process scales linearly with the number of configurations compared to cubic time complexity in Gaussian processes-based BO methods. Hence, BOHB chooses TPE for building hyperparameter models. However, one should note that there are no theoretical guarantees of TPE, unlike GP-based models.

4.2 The Algorithm

We now introduce the BOHB algorithm and begin by discussing various design choices in the paper. BOHB takes advantage of the BO's strong final performance by guiding the selection process of configurations before each HB iteration. The BO part is handled by a variant of the previously introduced Parzen Estimator (TPE) with a single multi-dimensional KDE compared to a product of univariate distributions used in TPE. This is to better handle the interaction effects in the input space.

Algorithm 2 Pseudocode for sampling BOHB

input: observations D, fraction of random runs ρ, percentile q, number of samples N_s, minimum number of points N_{min} to build a model, and a bandwidth factor b_w
output: next configuration to evaluate
if $rand() < \rho$ **then**
  return random configuration
end if
$b = argmax D_b : |D_b| \geq N_{min} + 2$
if $b = \emptyset$ **then**
  return random configuration
end if
fit KDEs
draw N_s samples according to $l'(x)$

return sample with highest ratio $l(x)/g(x)$

In this section, we will mostly focus on the BO based sampling strategy in BOHB since the HB part mostly remains intact. BOHB takes as input the number of observations D, fraction of random runs ρ, percentile q, number of samples N_s, the minimum number of points N_{min} to build a model, and bandwidth factor b_w.

We proceed by providing some intuition behind each of the main steps. The algorithm begins by uniformly sampling a fraction of configurations ρ at random. The idea behind this is two-fold. First, the random configuration space provides an opportunity to explore the global space. Secondly, this ensures that for every Hyperband based SH run consuming a budget of $s_{max} + 1$, the random search evaluates ($\rho^1 \cdot (s_{max} + 1)$) times as many configurations on the same budget. Consequently, BOHB is guaranteed to converge eventually, even if it is this factor times slower than random search.

The second step of the algorithm looks for the largest budget b for which enough observations are available. This ensures that the BO-based sample selection is always optimized on the biggest budget possible at any optimization time-step. Since smaller budgets can be misleading (as seen in the SH example), this helps them avoid any wrong assessments made based on smaller budgets. If there are no such budgets, such as if enough evaluations are not there, they simply return a random configuration in step three.

To understand the fourth step, it is useful to note that a minimum number of data points are required to fit KDEs. As discussed in the TPE introduction, the algorithm partitions data points in the space based on a threshold value. The minimum number of points N_{min} is usually initialized as $d + 1$, where d is the number of hyperparameters. In such a scenario, the algorithm needs to wait for enough points $N_b = |D_b|$, such that the number of observations for budget b satifies $q \cdot N_b \geq N_{min}$. Here, q refers to the percentile and defines the percentage of "good" and "bad" points in the algorithm. For instance, if q is set such that

the model identifies the top 15% values as "good" points, then for any given budget b, the algorithm needs at least 15% of the observations $(q \cdot N_b)$ to be equal to the minimum number of points N_{min}. To mitigate this shortcoming and reduce the waiting time, the authors initialize the model with $N_{min} + 2$ random configurations in line 3 and define:

$$\begin{aligned} N_{b,l} &= max(N_{min}, q \cdot N_b) \\ Nb, g &= max(N_{min}, N_b N_{b,l}) \end{aligned} \tag{9}$$

These values define the number of good and bad configurations, respectively. This choice of values ensures that both models have minimal overlap in terms of datapoint selection when only a fraction of observations are available. Finally, the model fit KDEs according to Equation 8 and 9.

For step 5, the authors sample N_s points from $l^{'}(x)$ which is basically same as $l(x)$ obtained in the last step but multiplied by the bandwidth factor b_w. As discussed during the introduction of KDEs, bandwidth controls the smoothness and spread of the kernel function. With large enough b_w, the model encourages exploration around a wider region with several promising configurations. The authors claim that this promotes convergence, especially in the latter stages of optimization.

Finally, the model samples the point with the highest ratio of $l(x)/g(x)$ for the final step. This, in turn, maximizes the expected improvement (EI).

4.3 Parallelization

One key advantage of BOHB comes from the massive parallelization offered by the algorithm, The method leverages both TPE and HB to ripe the combined result of parallelization. HB naturally offers ample opportunities of parallelism since different iterations of SH can be run concurently. Similarly, with each SH run, each configuration can start evaluation simultaneously. To obtain similar advantages from TPE, the authors limit the number of samples to optimize EI. The method follows a very simple approach to parallelize the algorithm. It starts with the first SH run and allocates as many worker threads as the method requires. Then, they sample configurations with the algorithm as discussed above either all workers are busy or enough observations are available.

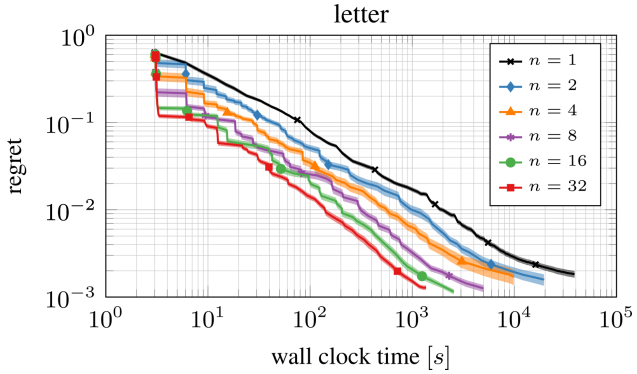


Figure 11: Performance of BOHB with different number of parallel workers for 128 iterations [2]

As shown in Figure 11, the speedup for two and four workers is close to linear. For more workers it

becomes sub-linear. The speedup for one vs. 32 workers is roughly 15 (2000s/130s).

4.4 Experimental Results

The paper evaluates BOHB over a wide range of machine learning algorithms including traditional machine learning algorithms such as SVM, Feed-forward neural network, reinforcement learning and even Bayesian Optimization. In general, the method outperforms all the other baselines and obtains the most optimal set of hyperparameter configurations in most cases.

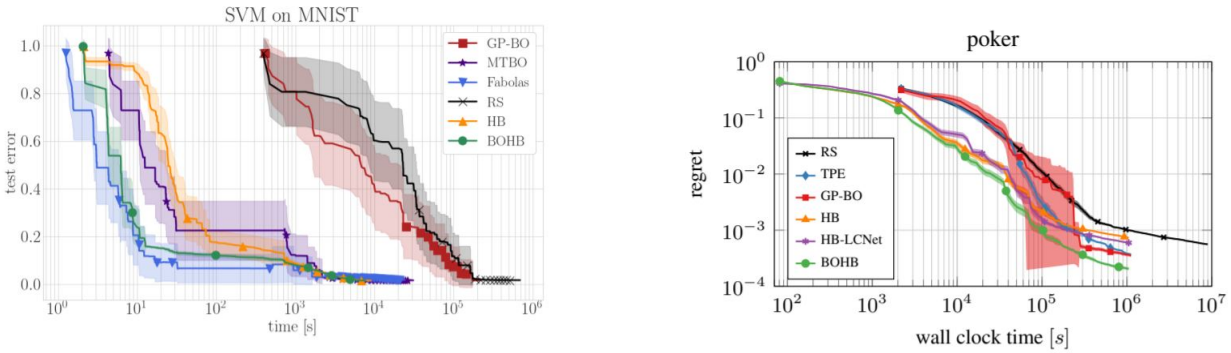


Figure 12: Left: Comparison on the SVM on MNIST surrogates and Right: Performance over Feed-forward neural network [2]

As can be observed in Figure 12 left, BOHB was compared against other baselines for the optimization of the regularization and the kernel parameter of an SVM on MNIST surrogate. FABOLAS performed relatively faster than other methods, while BOHB follows closely and achieves the best configurations. For Figure 12 right, BOHB was evaluated on the six hyperparameters (initial learning rate, batch size, dropout, exponential decay factor for learning rate) controlling the training process of a feed-forward neural network over six datasets. Here, HB initially performed better than vanilla BO methods. However, as discussed earlier, with larger budgets, TPE and BO catch up. Overall, BOHB performs the best achieving the same final score as HB roughly 100 times faster.

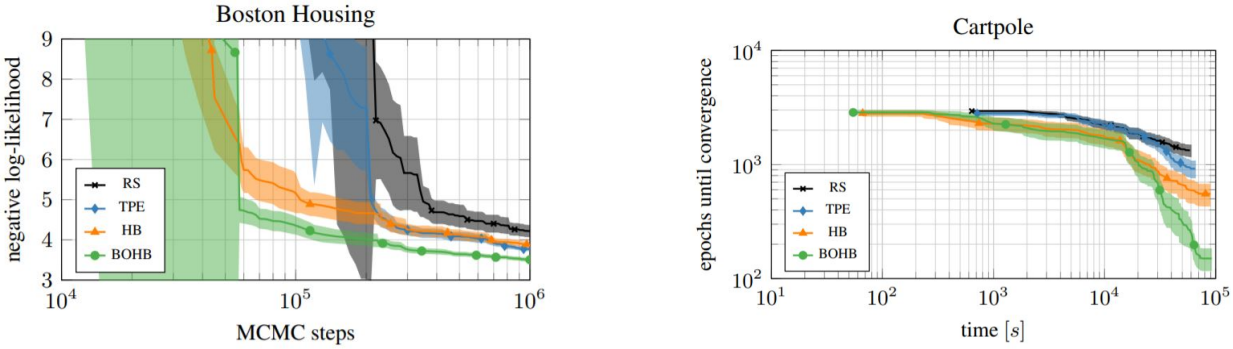


Figure 13: Left: Optimization of 5 hyperparameters of a Bayesian neural network trained with SGHMC on the Boston dataset and Right: Hyperparameter optimization of 8 hyperparameters of PPO on the cartpole task [2]

The prove the efficacy of the method over a wide range of ML algorithms. For this, authors perform two other interesting experiments. As can be seen in Figure 13, they optimize the hyperparameters and architecture of a two-layer fully connected Bayesian neural network trained with MCMC sampling. BOHB converged much faster than both HB and TPE even though HB performed well in the initial runs. Similarly, they also optimize the eight hyperparameters of proximal policy optimization (PPO) to learn the cartpole swing-up task. Here again, they observe better configurations with BOHB.

5 Conclusion & Future Directions

We discuss two advanced hyperparameter optimization methods, FABLOS and BOHB, in this report. FABOLAS exploits cheaper fidelities of the objective function by performing experiments on a subset of the training data and extrapolates the results on the full dataset. The method works well in practice, both in terms of good anytime and overall model performance. However, due to its reliance on the GP-based BO approach, it still suffers from cubic time complexity. BOHB, on the other hand, combines both Bayesian Optimization and Hyperband to get the best out of the two worlds. They rely on HB to determine the number of configurations to evaluate with an appropriate fixed budget and leverage BO to select configurations at the start of each HB iteration. The algorithm performs well across various benchmarks and different machine learning settings. Further, we study the efficiency of hyperparameter optimization of FABOLAS compared to Random Search and Grid Search over Fashion MNIST dataset for the LeNet5 neural network.

As future work, we plan to investigate the role of these HPO methods on more challenging experimental setups such as data-free Knowledge Distillation (KD). Data-free KD is the process of transferring knowledge from a trained neural network to a non-trained one in the absence of training data. As the approach is agnostic of the training dataset, finding optimal hyper-parameters is a non-trivial process. We may model the correlation between the training losses used in data-free KD with the real test accuracy based on proxy-publicly available datasets using Gaussian Processes. Based on these patterns, we may infer the optimal set of hyperparameters for other unknown datasets.

References

[1] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[2] Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale, 2018.

[3] Kevin Jamieson and Ameet Talwalkar. Non-stochastic best arm identification and hyperparameter optimization, 2015.

[4] Aaron Klein, Stefan Falkner, Simon Bartels, Philipp Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets, 2017.

[5] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization, 2018.

[6] Kevin Swersky, Jasper Snoek, and Ryan Prescott Adams. Multi-task bayesian optimization. 2013.

[7] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.