

Adaptive Bayesian contextual hyperband: A novel hyperparameter optimization approach

Lakshmi Priya Swaminatha Rao, Suresh Jaganathan

Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering, Chennai, Tamil Nadu, India

Article Info

Article history:

Received Sep 28, 2023

Revised Oct 17, 2023

Accepted Oct 29, 2023

Keywords:

Bayesian algorithms

Hyperband

Hyperparameter optimization

Machine learning models

Multi-armed bandits

ABSTRACT

Hyperparameter tuning plays a significant role when building a machine learning or a deep learning model. The tuning process aims to find the optimal hyperparameter setting for a model or algorithm from a pre-defined search space of the hyperparameters configurations. Several tuning algorithms have been proposed in recent years and there is scope for improvement in achieving a better exploration-exploitation tradeoff of the search space. In this paper, we present a novel hyperparameter tuning algorithm named adaptive Bayesian contextual hyperband (Adaptive BCHB) that incorporates a new sampling approach to identify best regions of the search space and exploit those configurations that produce minimum validation loss by dynamically updating the threshold in every iteration. The proposed algorithm is assessed using benchmark models and datasets on traditional machine learning tasks. The proposed Adaptive BCHB algorithm shows a significant improvement in terms of accuracy and computational time for different types of hyperparameters when compared with state-of-the-art tuning algorithms.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Lakshmi Priya Swaminatha Rao

Department of Computer Science and Engineering, Sri Sivasubramaniya Nadar College of Engineering

Chennai, Tamil Nadu, India

Email: lakshmipriyas@ssn.edu.in

1. INTRODUCTION

Hyperparameters (HPs) allow for a great deal of configuration of machine learning (ML) algorithms. Because the hyperparameter values frequently have considerable effects on the model's architecture, complexity, behavior, speed, and other features, it is important to choose them carefully in order to achieve the optimum results. Choosing these values through human trial and error methods ends up being time-consuming, frequently biased, error-prone, and computationally irreproducible. Yet, machine learning experts must adjust a number of hyperparameters to enhance task accuracy and minimize training resource consumption. Examples include the decision about the deep neural network (DNN) model architecture, the training dataset, the optimization technique, the optimization algorithm's hyperparameters (such as learning rate and momentum), and the training time budget. Optimization of hyperparameters started gaining momentum since the beginning of 1990s [1], and it has been known that specific hyperparameter settings tend to work better for various datasets. Contrarily, the ability of hyperparameter optimization (HPO) to modify general-purpose pipelines for usage in particular application domains is a very recent realization. It is now generally acknowledged that optimized hyperparameters outperform the default settings provided by the majority of machine learning frameworks. It is preferable to outsource HPO to appropriate algorithms and machines to improve efficiency and ensure reproducibility. This is because the mathematical formal-

sation of HPO is essentially a black box optimization, which is frequently in a higher-dimensional environment. The primary objective of automated machine learning (AutoML) is therefore to automatically configure these hyperparameters to improve performance [2]-[5] thereby saving precious execution time. In particular, deep neural networks that depend on a variety of hyperparameter decisions about their architecture, optimization, and regularization benefit the most from automated hyperparameter search. Hyperparameter optimization using AutoML techniques have several important advantages [6], [7] which includes: i) Reducing the human efforts by making the entire hyperparameter optimization process automated; ii) Tailoring the HPO process to suit the needs of the problem at hand; iii) Improving the reproducibility of experiments as it removes the need for manual search. Several HPO algorithms have been proposed in recent years and the efficiency of the algorithms depends on several factors including how the initial hyperparameter search space is defined and the exploration of such space. This paper aims to

- i) formulate a novel algorithm for hyperparameter tuning to achieve a good trade-off between exploration and exploitation of the hyperparameter search space. The algorithm does so by exploring new regions in the hyperparameter search space for better performing configurations by including the observations seen till time step t and exploiting the best configurations with more resources.
- ii) present a new sampling technique that creates a sample space by including equal number of the historical observations and random hyperparameter configurations based on an adaptive threshold for loss.
- iii) select only the best performing configurations in every iteration and diverting the resources only to these configurations thereby exploiting the hyperparameter search space.

This research proposes a novel hyperparameter tuning procedure called adaptive Bayesian contextual hyperband (Adaptive BCHB) along with an innovative sampling strategy to accomplish the aforementioned objectives. The rest of the paper is organized as follows: section 2 focuses on the recent literature on hyperparameter tuning, their challenges and applications. Section 3 explains the hyperparameter tuning process and the proposed tuning algorithm and the sampling technique. The experimental setup and results are shown in section 4 with conclusion in section 5.

2. RELATED WORK

Hyperparameter tuning is an important step in building an ML model for any application. Choosing the right hyperparameters will help in reaching a good accuracy of the model in less training time. There are several tuning algorithms with different strategies that has evolved through the years [1], [8], [9]. The hyperparameter search space is defined and the algorithms differ in how this space is navigated. Grid search (GS) [10] is a brute-force strategy which explores the search space in a grid and does an exhaustive search of all hyperparameters in it using a predefined step size. This method is popularly used when the number of hyperparameters is small. Alternatively, when the number of hyperparameters is more, grid search becomes computationally expensive as the number of evaluations grows exponentially and the result of the algorithm solely depends on the grid and step size. Random search (RS) [11] navigates the search space and selects configurations randomly between the specified lower and upper bounds for the hyperparameters. This ensures a fair exploration of the search space. As each hyperparameter configuration is selected independently of the others this algorithm can be parallelized [10].

To overcome the limitations of RS and GS, Bayesian concepts were included in the tuning algorithms to identify and concentrate on well-performing regions thereby reaching the optimum earlier [12]. Bayesian strategies [13] uses a prior and posterior distribution representing the search space and for every iteration, the posterior is updated based on the prior and current evidence. The iterations are sequential by nature. Some works like [14], [15] have achieved parallelism by dividing the search space into subspaces and applying a simpler tuning algorithm to find the best performing regions and then applying the intended tuning algorithm in the subspace containing the best region. There are several variations of Bayesian strategies based on the surrogate models used. Some commonly used surrogate models are Gaussian processes (GP), Tree Parzen Estimators (TPE) and random forest (RF) making the Bayesian optimization (BO) based tuning algorithms as BO-GP, BO-TPE and sequential model based algorithm configuration (SMAC) respectively [16]. Unlike BO-GP that is mainly used for hyperparameters that are continuous, SMAC supports discrete, continuous, categorical, and conditional hyperparameters. BO-TPE is tree based and naturally supports conditional hyperparameters.

To limit the use of resources, a well-defined budget (the algorithm's number of iterations or running time) may be set before the start of the tuning process. This budget can be utilized effectively for better perform-

ing configurations. Bandit based algorithms [17] are based on this strategy and two commonly used algorithms are 1) Successive halving 2) Hyperband [18]. Successive halving uniformly divides its budget among all the configurations selected. Hyperband uses successive halving as a subroutine but dynamically determines the number of configurations to assess and the budget for every iteration. It starts with a predefined number of configurations with a limited budget and finishes with a better performing configuration in the group which is run with the maximum budget. Bayesian optimization hyperband (BOHB) [19] is a technique that incorporates combines Bayesian techniques in the Hyperband algorithm. BOHB replaces the random search used in Hyperband to sample the configurations from the search space with BO which selects the configuration based on the surrogate model built. This results in higher accuracy in a less amount of time than successive halving and also Hyperband. Studies [20] have been conducted that proves BOHB surpasses in performance, many other optimization techniques when tuning traditional machine learning models like support vector machines (SVM) and also deep learning (DL) models.

Other than Bayesian algorithms, several different class of algorithms like metaheuristic algorithms like genetic algorithms (GA) [21], particle swarm optimization (PSO) are also used for hyperparameter tuning. There are several hybrid methodologies based on the above-mentioned commonly used algorithms that are used in many ML and DL models. These hybrid methods make use of the best of the techniques they use while avoiding their limitations. One such method discussed in [22] has been shown to work well for convolutional neural networks. A similar hybrid method based on PSO is used in autoencoders [23], [24]. Differential evolution (DE) strategies are used in [25] but the results show that they perform better with smaller datasets. DE algorithms are computationally expensive [26] and can be afforded only with the availability of huge computational resources [27].

3. METHOD

The hyperparameter tuning process is iterative and accepts the complete space of hyperparameters as input. In our earlier work [28], we had formulated an algorithm named Bayesian contextual hyperband (BCHB) that incorporates a surrogate model and an acquisition function to select the next best configuration to run based on the observed loss values in the previous iterations. An initial surrogate model constructed using default hyperparameters forms the basis of the algorithm. The inner loop of the algorithm (see BCHB algorithm of [28]) executes r_i times (the number of resources allocated for a configuration) for each iteration i to update the surrogate model with the next-best hyperparameter configuration and associated loss value. The acquisition function S is used to acquire the best hyperparameter which is denoted by h^* . The history of the best configurations (HT) seen thus far is kept as the pair $(configuration, loss)$, and the surrogate model is revised with this past knowledge. This configuration h^* is assessed with the original loss function. Like BOHB, the *get.hyperparameter_configuration(n)* function in this algorithm randomly selects n configurations from the search space at the beginning of each bracket. Some interesting points to note in BCHB that has scope for improvement are: i) For the start of each bracket of BCHB, random sampling is used to select n new configurations from surrogate model which involves no exploitation of the observed values ii) The number of configurations from second iteration of successive having in each bracket is fixed even if the configuration performs poorly iii) The number of resources allotted to a configuration in each iteration is also fixed leading to wastage of computational resources for poorly performing configurations.

3.1. Adaptive Bayesian contextual hyperband (Adaptive BCHB)

To address the above concerns and improve the performance of tuning, we propose an improved version of BCHB named adaptive Bayesian contextual hyperband (Adaptive BCHB) which is given in Algorithm 1. The algorithm starts with defining R , the maximum number of resources that can be allotted to a single configuration and a surrogate model M_0 built with default settings of the model hyperparameters. The hyperparameter search space H is formed and minimum threshold for the validation loss is initialized as T . The algorithm uses an acquisition function S that aids in selecting next best hyperparameters for the following iterations after eliminating the poor performers based on a control parameter η (set to default = 3) [19]. The algorithm uses successive halving procedure of Hyperband and finally returns the optimal hyperparameter configuration. The key difference between BCHB and the proposed algorithm is that the latter keeps track of the history of observed configurations as HT . This history is used for sampling new configurations from the search space as in Algorithm 2 for the subsequent iterations.

Algorithm 1 Adaptive Bayesian contextual hyperband (Adaptive BCHB)**Input:** R, M_0, H, S, T, η **Output:** Optimal hyperparameter configurationInitialize $HT \leftarrow \emptyset$ and $s_{max} = \log_\eta(R)$ **for** $s \in \{s_{max}, s_{max}-1, \dots, 0\}$ **do** **if** $s = s_{max}$ **then** $HP \leftarrow \text{Randomly sample } n \text{ configurations from } H$ **else** $HP \leftarrow \text{get_hyperparameter_configuration}(n)$ **end if**Calculate $n = \lceil \frac{B\eta^s}{R(s+1)} \rceil, R\eta^{-s}$ **for** $i \in 1, 2 \dots s$ **do** **if** $i = 0$ **then** $n_i = \lfloor n\eta^{-i} \rfloor$ and $r_i = r\eta^i$ **else** $n_i = \text{count}(HP)$ and $r_i = \lceil \frac{R}{n_i} \rceil$ **end if** **for** $j \leftarrow 1$ to r_i **do** $h^* = \text{argmax}_h S(h, M_{j-1})$ Evaluate $f(h^*)$ for one additional step $HT \leftarrow HT \cup (h^*, f(h^*)_{k+1})$ Fit a new model M_j to HT **end for** $L = \text{run_then_return_val_loss}(h, r_i) : h \in HP$ $NewHP \leftarrow \emptyset$ **for all** $h \in HP$ **do** **if** $L[h, r_i, l_i] < \text{set_threshold}(T)$ **then** Add h to $NewHP$ **end if** **end for** $HP = NewHP$ **end for****end for****return** Optimal Hyperparameter Configuration

The contributions in the improved algorithm Adaptive BCHB are as follows:

- i The algorithm incorporates a novel sampling technique that explores the best regions of the original search space obtained from the history of best configurations observed. The optimal samples for the next iteration of the algorithm are identified using a reward function.
- ii A threshold for validation loss is used to select the number of configurations that furthers through the successive halving iterations. This threshold is updated dynamically to select more potential candidates from the search space rather than being stuck with a local optimum.
- iii There exists an upper bound for the number of times a hyperparameter configuration is selected for execution. This is to include some level of randomness in the selection process enabling new configurations to be pulled resulting in the balance between exploration and exploitation of the search space.

Algorithm 2 shows the new sampling approach that uses a search space H_{new} formed using sets of randomly sampled configurations from the original search space and the best configurations in history. A hyperparameter configuration h_i can be selected for the first time or for the $(k+1)^{th}$ time assuming it is already selected k times. The number of times the configuration is pulled is limited to N to allow other potential candidates a chance to run. When a configuration h_i uses k resources it produces an output $z_{i,k}$ (model performance). The aim is to find the maximum z using a minimum number of resources. To achieve this, a reward rw_i is attached to the configuration h_i based on its performance [29]. Now maximizing $z_{i,k}$ is nothing but maximizing the reward rw_i . The reward function is updated every time based on the observed

values. The probability density function of reward $rw_{i,k}$ is given in (1).

$$F(rw_{i,k}) = F([y_{i,k} - y^*]^+) = \begin{cases} \mathcal{N}(\mu_{i,k} - y^*, \sigma_{i,k}), & \text{if } y_{i,k} > y^* \\ \Phi(y^*; \mu_{i,k}, \sigma_{i,k}) & \text{if } y_{i,k} \leq y^*, rw_{i,k} = 0 \end{cases} \quad (1)$$

where $\Phi(x; \mu, \sigma) = \int_{-\infty}^x f(x) dx$ is the cumulative distribution function of the normal distribution $f(x|\mu, \sigma)$, y^* is the current best observation and $[x]^+$ is defined as $\max(0, x)$.

Algorithm 2 Sampling in adaptive BCHB

Input: H, HT, n

Output: Top n hyperparameter configurations

$RD \leftarrow$ Set of n hyperparameter configurations randomly sampled from H

$H_{old} \leftarrow$ Set of n hyperparameter configurations sampled from HT

$H_{new} \leftarrow RD \cup H_{old}$

for all $h_i \in RD$ **do**

$k = 0$

end for

for $h_i \in H_{new}$ **do**

randomly sample $k \in U[k+1, N]$

sample a reward rw_i from distribution $F(rw_{i,k})$

end for

return top n configurations with the best rewards

3.1.1. Dynamic threshold for validation loss

The *run_then_return_val_loss* (h, r_i) function of Algorithm 1 returns the loss value obtained by running the model with r resources configured for hyperparameters h in round i . Based on these losses, the *top_n*() method identifies the top n configurations. Not all of the top-performing candidates reach the lowest cutoff T and advance to the following cycle. The algorithm's ability to locate the ideal hyperparameter in a shorter period of time is significantly improved by eliminating the underperforming candidates in each iteration and allocating resources effectively in the following iteration. The threshold for the current round is set using the *set_threshold*(T) function which returns a new threshold T_i which is the average of the threshold at round $i - 1$ and the best value for loss (minimum) seen in the current round i as given in (2).

$$T_i = (T_{i-1} + T_{best}) / 2 \quad (2)$$

$$L[h, r_i] < T_i \quad (3)$$

The algorithm guarantees that the loss value can be modified dynamically for each iteration, depending on the application, by averaging the threshold. $L[h, r_i]$ denotes the loss for a configuration h in HP running on r_i resources. The configurations that meet the requirement stated in (3) are the ones to go to the subsequent halving iteration $i + 1$. This allows the threshold to change every round and the algorithm moves towards the best performing region in the search space much faster.

3.1.2. Adaptive budget

In the BCHB algorithm, each hyperband bracket has a predetermined minimum budget r that is chosen at the beginning of the bracket. At least once in the bracket, all n configurations are run using r resources. The input η and the bracket number s govern both n and r . The most exploratory bracket, $s = s_{max}$ has the smallest budget because it focuses on examining the search space for n distinct configurations. The bracket with the greatest potential for exploitation, $s = 0$, runs configurations for a budget R , which corresponds to the maximum number of resources that can be allotted to a configuration. The number of configurations in each bracket s and the budget's subsequent halving for iteration i are determined by η and i .

In Adaptive BCHB, the number of configurations that advance to iteration $i + 1$ is dynamic and fewer than or equal to n (of *top_n*()). This is achieved by setting the threshold and selecting the best configurations

that meet the threshold. However, the number of configurations chosen from HP and the resources given to those configurations are determined using (4) for the first iteration of each bracket, where $i = 0$.

$$n_i = \lfloor n\eta^{-i} \rfloor \quad r_i = r\eta^i \quad (4)$$

This is the same technique adopted in BCHB. It allows the initial iteration to explore the search space using the greatest possible number of combinations while spending the least number of resources possible. Additionally, for following iterations $i + 1$, the updated hyperparameter set HP defines the configurations that are chosen with the aid of the threshold T_i . (5) provides the number of hyperparameters n_i in these iterations, which is just the set HP's count.

$$n_i = \text{count}(HP) \quad (5)$$

It is only sensible to modify the minimal resources for each selected configuration as n_i is now less than or equal to n of $\text{top-}n()$. n_i now governs the allocation of resources rather than η . In other words, the number of resources provided to each configuration is adapted based on the number of configurations that advance to the next round. As a result, (6) is now used to determine the quantity of resources r_i in the iteration $i + 1$.

$$r_i = \left\lceil \frac{R}{n_i} \right\rceil \quad (6)$$

The Adaptive BCHB algorithm thus improves the exploitation of resources by allotting them to only the best performing candidates. This strategy eliminates the poor-performing configurations much early in the iterations and helps the algorithm to converge to the optimal configuration faster.

4. RESULTS AND DISCUSSION

The performance of the proposed algorithm is assessed for the machine learning tasks using the benchmark datasets as given in Table 1. The rationale for selecting these models is that they each feature a distinct type of hyperparameter and are most effective for both classification and regression tasks. SVM has conditional hyperparameters, RF and neural network (NN) each have multiple hyperparameters of various types, whereas K-nearest neighbor (KNN) has just one discrete parameter.

Table 1. Models, Datasets and Tuners used in the experiments

Tasks	Models Used	Datasets	Tuners Used
Classification		MNIST – handwritten digit dataset	Random Search (RS) [11]
		CIFAR-10 – objects dataset	Grid Search (GS) [10]
Regression	K-Nearest Neighbor (KNN)	Boston Housing Dataset	Bayesian Optimization with
	Support Vector Machines (SVM)		Tree Parzen Estimators (BO-TPE) [16]
	Random Forest (RF)		Bayesian Contextual Bandits (BCO) [29]
	Feed-Forward Neural Network (NN)		Hyperband (HB) [18]
			Bayesian Optimization Hyperband (BOHB) [19]
			Bayesian Contextual Hyperband (BCHB) [28]
			Adaptive BCH

Each experiment uses 5-fold cross-validation on the chosen HPO methodologies. While mean squared error (MSE) is employed for the regression job using the Boston Housing dataset, accuracy is the assessment metric for the classification task using the Modified National Institute of Standards and Technology database (MNIST) and Canadian Institute for Advanced Research, 10 classes (CIFAR-10) datasets. The entire execution time of each method with 5-fold cross validation is referred to as computational time, which is also employed as a statistic. Python 3.6 is used for all experiments on an Intel i7-8559U machine with 12 GB of RAM. Table 2 lists the machine learning tasks, models, selected hyperparameters, and their domain. All of the hyperparameters given in Table 2 for each of the experiments are included in the hyperparameter search space H .

Table 2. Model hyperparameters and their search space

Task	Model	Hyperparameters	Hyperparameter type	Hyperparameter domain
Classification	KNN	Number of neighbors	Discrete	[1,20]
		Regularizer	Continuous	[0.1,50]
	SVM	Kernel Functions	Categorical	['linear', 'poly', 'rbf', 'sigmoid']
		n_estimators	Categorical	[10,100]
	RF	max_depth	Discrete	[5,50]
		min_samples_split	Discrete	[2,11]
		min_samples_leaf	Discrete	[1,11]
		max_features	Discrete	[1,64]
		criterion	Categorical	['gini', 'entropy']
	NN	Number of hidden layer neurons	Discrete	[4,8,16,32]
		Dropout rate	Discrete	[0.5,0.8]
		Optimizer	Categorical	['Adam', 'SGD', 'RMSProp']
		Learning rate	Continuous	[0.01,1.0]
Regression	KNN	Number of neighbors	Discrete	[1,20]
		Epsilon	Continuous	[0.001,1]
	SVM	Regularizer	Continuous	[0.1,50]
		Kernel Functions	Categorical	['linear', 'poly', 'rbf', 'sigmoid']
	RF	n_estimators	Discrete	[10,100]
		max_depth	Discrete	[5,50]
		min_samples_split	Discrete	[2,11]
		min_samples_leaf	Discrete	[1,11]
		max_features	Discrete	[1,13]
		criterion	Categorical	['mse', 'mae']
	NN	Number of hidden layer neurons	Discrete	[4,8,16,32]
		Dropout rate	Discrete	[0.5,0.8]
		Optimizer	Categorical	['Adam', 'SGD', 'RMSProp']
		Learning rate	Continuous	[0.01,1.0]

4.1. Performance of Adaptive BCHB with popular tuning algorithms

Tables 3-5 list the results of the tuning algorithms used on the models. The proposed algorithm is compared with benchmark algorithms in terms of accuracy and computational time. The models are also run with default hyperparameters [20] to show in comparison the increase in performance after tuning.

From the results, it is evident that in terms of both accuracy and time, Adaptive BCHB outperforms all other tuning algorithms in all the cases. It can handle all types of hyperparameters (discrete, continuous, and categorical) well and produces the best results in comparison. The time taken to train the models with the default hyperparameters is less than the algorithms as there are no tuning done and these values are included in the tables to show how each model perform on its own without any hyperparameter tuning and to set the baseline. Classification tasks involving MNIST and CIFAR datasets have images as input and hence computational time is slightly more than the regression task. Grid search performs poorly as expected because of exploring the search space in a fixed grid and not exploiting the best regions in the space. Though the model performance of GS is closer to RS, it takes twice as much time as RS to complete execution. The Bayesian method BO-TPE performs better than grid and random search but fails to compete with bandit influenced Bayesian methods. Hyperband and BOHB compete closely in terms of model performance but there is a significant improvement in BOHB with respect to computational time.

The proposed algorithm, Adaptive BCHB does well with computational time (better than BCHB) since the poorly performing hyperparameter candidates in each round of the algorithm are discarded and their resources diverted to better performers. This in turn helps the algorithm to exploit only the best performing regions and limits exploration which saves time. The model performance of Adaptive BCHB is 7% more than the default hyperparameter settings for random forest as shown in Table 4. The interesting point to note here is that Adaptive BCHB performs well for all tasks using the neural network model in terms of both model performance and time. It is 1.5 times faster than BOHB when used with NN as shown in Table 5. Among all the Bayesian methods, Adaptive BCHB has performed well for all three tasks on three benchmark datasets with different types of hyperparameters.

Table 3. Accuracy and computational time for classification task using MNIST dataset

Tuning Algorithm	Accuracy (%)				Computational time (sec)			
	KNN	SVM	RF	NN	KNN	SVM	RF	NN
Default Hyperparameters	88.42	85.45	83.98	86.97	1.69	2.78	3.45	25.23
Random Search	89.46	87.78	86.78	86.88	6.32	17.89	10.23	33.34
Grid Search	90.98	88.57	86.24	86.22	8.67	25.78	30.34	56.34
BO-Tree Parzen Estimator	90.29	89.44	86.89	88.34	4.81	15.67	14.23	39.24
Bayesian Contextual Optimization	91.78	90.12	88.96	89.45	4.92	11.76	12.19	31.21
Hyperband	91.34	90.47	88.90	89.78	3.67	13.89	7.45	29.56
BOHB	92.45	90.49	89.23	91.06	3.72	9.45	8.38	27.45
BCHB	94.67	93.26	93.55	93.34	3.10	8.56	7.23	15.34
Adaptive BCHB	97.01	97.98	95.88	96.77	2.09	6.26	5.56	13.67

Table 4. Accuracy and computational time for classification task using CIFAR-10 dataset

Tuning Algorithm	Accuracy (%)				Computational time (sec)			
	KNN	SVM	RF	NN	KNN	SVM	RF	NN
Default Hyperparameters	83.34	81.26	80.12	83.46	1.23	5.34	2.98	15.26
Random Search	84.88	83.56	82.25	86.14	5.45	19.34	11.87	36.98
Grid Search	85.46	84.89	82.91	86.25	8.50	23.16	20.66	76.24
BO-Tree Parzen Estimator	85.90	84.11	83.11	87.90	5.56	11.38	16.99	41.44
Bayesian Contextual Optimization	86.34	85.35	83.47	87.16	4.34	10.75	13.25	35.75
Hyperband	87.13	86.39	83.55	88.33	3.35	13.69	9.43	32.38
BOHB	88.16	87.34	84.78	89.77	3.71	10.20	9.24	29.74
BCHB	91.80	90.19	85.23	91.60	2.89	9.44	8.55	28.10
Adaptive BCHB	93.91	94.57	87.55	95.57	1.34	7.11	6.89	24.11

Table 5. Error and computational time for regression task using Boston Housing dataset

Tuning Algorithm	Accuracy (%)				Computational time (sec)			
	KNN	SVM	RF	NN	KNN	SVM	RF	NN
Default Hyperparameters	80.34	77.76	44.45	26.45	0.03	0.97	1.45	4.98
Random Search	76.67	74.47	43.46	23.98	1.35	1.21	5.47	6.45
Grid Search	76.22	74.11	42.25	23.67	2.15	3.35	12.45	13.24
BO-Tree Parzen Estimator	74.56	73.78	41.88	22.67	1.85	2.74	4.18	5.98
Bayesian Contextual Optimization	73.78	72.45	40.67	21.34	1.35	2.13	3.94	5.67
Hyperband	73.56	72.32	40.35	20.38	0.98	1.35	3.26	5.23
BOHB	72.89	72.18	40.32	19.35	0.84	1.11	3.85	4.97
BCHB	71.45	71.23	38.28	16.46	0.56	0.96	2.87	3.15
Adaptive BCHB	68.36	66.25	34.34	12.15	0.12	0.15	1.26	2.10

4.2. Performance comparison of BCHB and Adaptive BCHB with varying budgets

The performance of the proposed algorithm Adaptive BCHB is compared with its predecessor BCHB using the NN model and its hyperparameters. The model is run with all the three benchmark datasets mentioned earlier and the results are plotted in terms of test error with varying budgets (the number of iterations).

Figure 1 illustrates the error for the MNIST dataset, where Adaptive BCHB initially produces errors that are more similar to BCHB but gradually improves as the number of iterations increases. Due to the adoption of an appropriate starting cutoff, the test error obtained for the CIFAR-10 dataset during the early iterations when compared to BCHB is significantly different, as shown in Figure 2. In the subsequent iterations, as the threshold for validation loss is updated based on the previous iterations, the proposed approach converges with a lesser difference in the test error. The technique starts with a 17% drop in error for the Boston housing dataset (see Figure 3) and converges faster than it does for the other two datasets. Additionally, there is a sizable error difference between BCHB and Adaptive BCHB throughout the iterations of the algorithm. When the budget is 200 iterations for the MNIST dataset and 150 iterations for the CIFAR-10 and Boston housing datasets, the Adaptive BCHB method begins to converge.

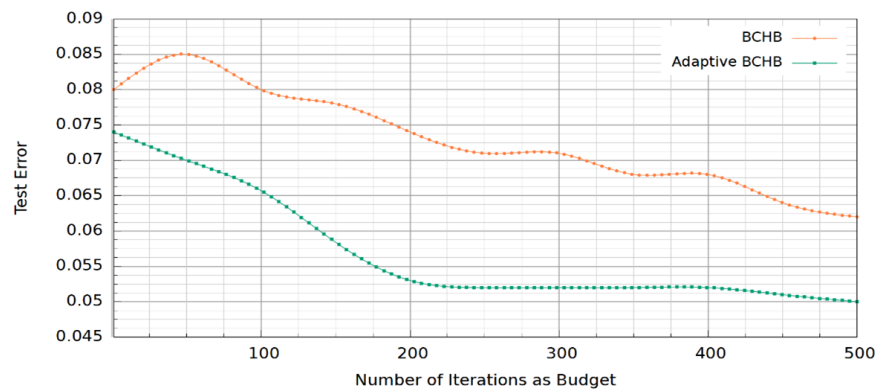


Figure 1. Varied budget analysis of Adaptive BCHB and BCHB on MINST dataset

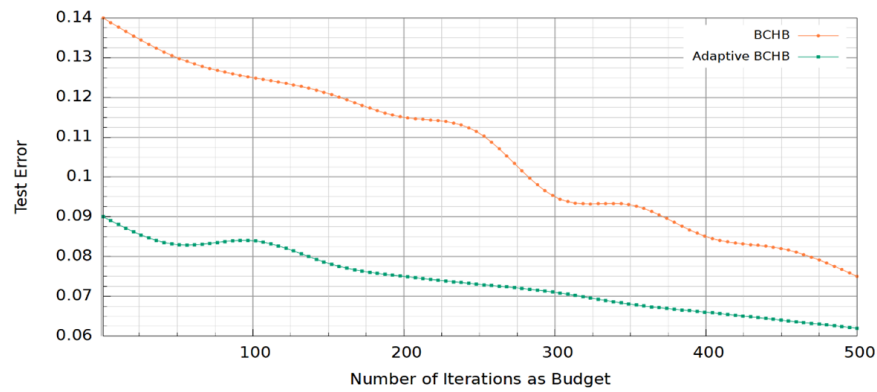


Figure 2. Varied budget analysis of Adaptive BCHB and BCHB on CIFAR-10 dataset

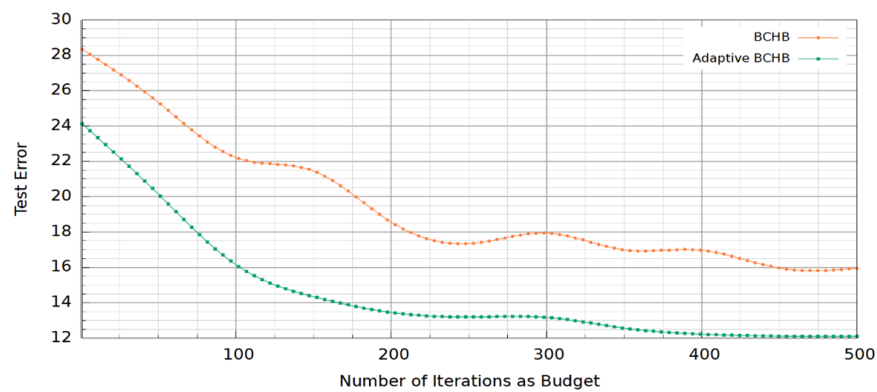


Figure 3. Varied budget analysis of Adaptive BCHB and BCHB on Boston dataset

5. CONCLUSION

Data analysis has gained attention in recent years due to the availability of computational resources. ML and DL models applied to these applications need to perform better to satisfy growing trends. Any good ML or DL model needs its hyperparameters to be set and manually tuning these hyperparameters is not only computationally expensive but time-consuming. Trial and error methods often are error prone. Automating the process of hyperparameter tuning hence becomes a necessity. Several hyperparameter tuning algorithms

have been developed in recent years but studies have shown that there is still scope for improvement when these algorithms are applied to real-time applications. In this paper, we have proposed a novel hyperparameter tuning algorithm which relies on Bayesian methods and multi-armed bandits. To prove the performance of the algorithm, the proposed algorithm is compared with other tuning algorithms that work on three benchmark datasets for classical machine learning tasks. The algorithm improves on the trade-off between exploration and exploitation of the hyperparameter search space. It explores new regions of the search space by limiting the number of times each configuration is pulled and at the same time it exploits the best performing hyperparameter configurations by allocating more resources in each iteration. i.e., the algorithm adapts to explore, and exploit based on the observed values at each time step. The results show that the proposed Adaptive BCHB algorithm performs well in terms of accuracy and time. It can be said with confidence that since the algorithm performs well for benchmark datasets and models, it can be scaled to adapt to any real-time application.





REFERENCES

- [1] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *Advances in neural information processing systems*, vol. 24, 2011, doi:10.5555/2986459.2986743.
- [2] N. Tran, J.-G. Schneider, I. Weber, and A. Qin, "Hyper-parameter optimization in classification: To-do or not-to-do," *Pattern Recognition*, vol. 103, p. 107245, 2020, doi:10.1016/j.patcog.2020.107245.
- [3] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for health-care," *Artificial intelligence in medicine*, vol. 104, p. 101822, 2020, doi: 10.1016/j.artmed.2020.101822.
- [4] G. Luo, "A review of automatic selection methods for machine learning algorithms and hyper-parameter values," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, pp. 1–16, 2016, doi:10.1007/s13721-016-0125-6.
- [5] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019, doi:10.1007/978-3-030-05318-5.
- [6] L. Vaccaro, G. Sansonetti, and A. Micarelli, "An empirical review of automated machine learning," *Computers*, vol. 10, no. 1, p. 11, 2021, doi:10.3390/computers10010011.
- [7] M.-A. Zöller and M. F. Huber, "Benchmark and survey of automated machine learning frameworks," *Journal of artificial intelligence research*, vol. 70, pp. 409–472, 2021, doi:10.1613/jair.1.11854.
- [8] G. Menghani, "Efficient deep learning: A survey on making deep learning models smaller, faster, and better," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–37, 2023, doi:10.1145/3578938.
- [9] M. J. Kochenderfer and T. A. Wheeler, *Algorithms for optimization*. MIT Press, 2019, doi:10.5555/3351864.
- [10] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur, M. Hardt, B. Recht, and A. Talwalkar, "A system for massively parallel hyperparameter tuning," *Proceedings of Machine Learning and Systems*, vol. 2, pp. 230–246, 2020, doi:10.48550/arXiv.1810.05934.
- [11] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. 10, pp. 281–305, 2012, doi:10.5555/2188385.2188395.
- [12] H. Alibrahim and S. A. Ludwig, "Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization," in *2021 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2021, pp. 1551–1559, doi:10.1109/CEC45853.2021.9504761.
- [13] S. Greenhill, S. Rana, S. Gupta, P. Vellanki, and S. Venkatesh, "Bayesian optimization for adaptive experimental design: A review," *IEEE access*, vol. 8, pp. 13 937–13 948, 2020, doi:10.1109/ACCESS.2020.2966228.
- [14] M. T. Young, J. D. Hinkle, R. Kannan, and A. Ramanathan, "Distributed bayesian optimization of deep reinforcement learning algorithms," *Journal of Parallel and Distributed Computing*, vol. 139, pp. 43–52, 2020, doi:10.1016/j.jpdc.2019.07.008.
- [15] T. Hinz, N. Navarro-Guerrero, S. Magg, and S. Wermter, "Speeding up the hyperparameter optimization of deep convolutional neural networks," *International Journal of Computational Intelligence and Applications*, vol. 17, no. 02, p. 1850008, 2018, doi:10.1142/S1469026818500086.
- [16] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimization," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019, doi:10.11989/JEST.1674-862X.80904120.
- [17] D. Guo, S. I. Ktena, P. K. Myana, F. Huszar, W. Shi, A. Tejani, M. Kneier, and S. Das, "Deep bayesian bandits: Exploring in online personalized recommendations," in *Proceedings of the 14th ACM Conference on Recommender Systems*, 2020, pp. 456–461, doi:10.1145/3383313.3412214.
- [18] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *The journal of machine learning research*, vol. 18, no. 1, pp. 6765–6816, 2017, doi:10.48550/arXiv.1603.06560.
- [19] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *International conference on machine learning*. PMLR, 2018, pp. 1437–1446, doi:10.48550/arXiv.1807.01774.
- [20] L. Yang and A. Shami, "On hyperparameter optimization of machine learning algorithms: Theory and practice," *Neurocomputing*, vol. 415, pp. 295–316, 2020, doi:10.1016/j.neucom.2020.07.061.
- [21] M. Munsarif, E. Noersasongko, P. N. Andono, and M. A. Soeleman, "The evaluation of convolutional neural network and genetic algorithm performance based on the number of hyperparameters for english handwritten recognition," *International Journal of Artificial Intelligence*, vol. 12, no. 3, pp. 1250–1259, 2023, doi:10.11591/ijai.v12.i3.pp1250-1259.
- [22] H. Cui and J. Bai, "A new hyperparameters optimization method for convolutional neural networks," *Pattern Recognition Letters*, vol. 125, pp. 828–834, 2019, doi:10.1016/j.patrec.2019.02.009.
- [23] Y. Sun, B. Xue, M. Zhang, and G. G. Yen, "An experimental study on hyper-parameter optimization for stacked auto-encoders," in *2018 IEEE congress on evolutionary computation (CEC)*. IEEE, 2018, pp. 1–8, doi:10.1109/CEC.2018.8477921.
- [24] C. Di Francescomarino, M. Dumas, M. Federici, C. Ghidini, F. M. Maggi, W. Rizzi, and L. Simonetto, "Genetic algorithms





- for hyperparameter optimization in predictive business process monitoring,” *Information Systems*, vol. 74, pp. 67–83, 2018, doi:10.1016/j.is.2018.01.003.
- [25] J. Han, C. Gondro, K. Reid, and J. P. Steibel, “Heuristic hyperparameter optimization of deep learning models for genomic prediction,” *G3*, vol. 11, no. 7, p. jkab032, 2021, doi:10.1093/g3journal/jkab032.
- [26] Y. Yoo, “Hyperparameter optimization of deep neural network using univariate dynamic encoding algorithm for searches,” *Knowledge-Based Systems*, vol. 178, pp. 74–83, 2019, doi:10.1016/j.knosys.2019.04.019.
- [27] B. Nakisa, M. N. Rastgoo, A. Rakotonirainy, F. Maire, and V. Chandran, “Long short term memory hyperparameter optimization for a neural network based emotion recognition framework,” *IEEE Access*, vol. 6, pp. 49 325–49 338, 2018, doi:10.1109/ACCESS.2018.2868361.
- [28] L. P. Swaminatha Rao and S. Jaganathan, “Intelligent short term traffic forecasting using deep learning models with bayesian contextual hyperband tuning,” *Computational Intelligence*, vol. 38, no. 6, pp. 2009–2034, 2022, doi:10.1111/coin.12554.
- [29] G. Sui and Y. Yu, “Bayesian contextual bandits for hyper parameter optimization,” *IEEE Access*, vol. 8, pp. 42 971–42 979, 2020, doi: 10.1109/ACCESS.2020.2977129.

BIOGRAPHIES OF AUTHORS



Lakshmi Priya Swaminatha Rao     received the Bachelor of Engineering degree in Information Technology from Bharathiar University, India and Masters of Engineering in Computer Science from Anna University, India in 2003 and 2005 respectively. Currently, she is an Assistant Professor in the Department of Computer Science and Engineering at SSN College of Engineering, India. She is currently pursuing her PhD under Anna University in the area of Data Analytics. Her research interests are in the areas of Big data analytics using probabilistic graphical models and various machine learning techniques. She can be contacted at email: lakshmipriyas@ssn.edu.in.



Suresh Jaganathan     an Associate Professor in the Department of Computer Science and Engineering has more than 20 years of teaching experience. He received his PhD in Computer Science from Jawaharlal Nehru Technological University, Hyderabad, M.E. Software Engineering from Anna University and B.E. Computer Science and Engineering, from Madurai Kamarajar University, Madurai. He has more than 30 publications in refereed International Journals and Conferences. Apart from this, to his credit he has filed 4 patents and written a book on *Cloud Computing: A Practical Approach for Learning and Implementation*, published by Pearson Publications. He is active reviewer in reputed journals (Elsevier - Journal of Networks and Computer Applications, Computer in Biology and Medicine) and also co-investigator for the SSN-NVIDIA GPU Education center. His areas of interest are Distributed Computing, Deep Learning, Data Analytics, and Machine learning. He can be contacted at email: sureshj@ssn.edu.in.