Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Extending the hypergradient descent technique to reduce the time of optimal solution achieved in hyperparameter optimization algorithms

Farshad Seifi[a] and Seyed Taghi Akhavan Niaki[b*]

[a]*Department of Industrial Engineering, Sharif University of Technology, Tehran, Iran*
[b]*Department of Industrial Engineering, Sharif University of Technology, P.O. Box 11155-9414, Azadi Ave, Tehran 1458889694, Iran*

C H R O N I C L E

Article history:
Received March 8 2023
Received in Revised Format March 29 2023
Accepted April 14 2023
Available online April, 14 2023

Keywords:
Hyperparameter optimization
Hypergradient descent
Multi-fidelity optimization
Bayesian optimization
Population-based optimization
Metaheuristic algorithm

A B S T R A C T

There have been many applications for machine learning algorithms in different fields. The importance of hyperparameters for machine learning algorithms is their control over the behaviors of training algorithms and their crucial impact on the performance of machine learning models. Tuning hyperparameters crucially affects the performance of machine learning algorithms, and future advances in this area mainly depend on well-tuned hyperparameters. Nevertheless, the high computational cost involved in evaluating the algorithms in large datasets or complicated models is a significant limitation that causes inefficiency of the tuning process. Besides, increased online applications of machine learning approaches have led to the requirement of producing good answers in less time. The present study first presents a novel classification of hyperparameter types based on their types to create high-quality solutions quickly. Then, based on this classification and using the hypergradient technique, some hyperparameters of deep learning algorithms are adjusted during the training process to decrease the search space and discover the optimal values of the hyperparameters. This method just needs only the parameters of the previous two steps and the gradient of the previous step. Finally, the proposed method is combined with other techniques in hyperparameter optimization, and the results are reviewed in two case studies. As confirmed by experimental results, the performance of the algorithms with the proposed method have been increased 36.62% and 23.16% (based on the best average accuracy) for Cifar10 and Cifar100 dataset respectively in early stages while the final produced answers with this method are equal to or better than the algorithms without it. Therefore, this method can be combined with hyperparameter optimization algorithms in order to improve their performance and make them more appropriate for online use by just using the parameters of the previous two steps and the gradient of the previous step.

1. Introduction

Machine learning algorithms have been prominently applied in various practical and scientific areas. As a result of such a prominent position, the demand for machine learning systems is growing. This outstanding position is due to machine learning algorithms' excellent performance and general use. However, it is a time-consuming and complicated process to build an efficient machine learning model, which includes determining the proper algorithm and obtaining an optimal architecture for the model. In general, two types of variables should be tuned for building a machine learning model. Firstly, the values of the variables identified during the model training process are determined by optimization techniques and are termed as parameters. Secondly, the variables that are not specified during the model training process should be presented as input to the model

* Corresponding author
E-mail: Niaki@Sharif.edu (S. T. A. Niaki)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)

doi: 10.5267/j.ijiec.2023.4.004

before implementing it (Liu, Wu, & Chen, 2022). These variables are termed hyperparameters. The hyperparameters can be discrete, continuous, conditional, or categorical, depending on the model (DeCastro-García, Muñoz Castañeda, Escudero García, & Carriegos, 2019; Yang & Shami, 2020).

In establishing an effective machine learning model, one of the vital components is tuning hyperparameters (Yao, Cai, Bu, & Chen, 2017). Hyperparameter optimization (HPO) is described as the process of adjusting the suitable values for a machine learning problem's hyperparameters. The usual practice to implement this process is using some approaches such as trial and error or the design of experiments manually. Nevertheless, these approaches are not practical anymore because the number of hyperparameters is high, the evaluations are time-consuming, there are interactions among nonlinear hyperparameters, and the models are usually complex (Yang & Shami, 2020). Thus, automated hyperparameters optimization is necessary, which has critical functions, with the following outcomes (Seifi, Azizi, & Niaki, 2021):

- The algorithm's good performance
- Effective use of parallel resources
- The reduced human effort needed for algorithm tuning
- Greater flexibility and robustness
- Higher algorithm scalability (Falkner, Klein, & Hutter, 2018)

HPO should deal with some challenges despite the advantages mentioned above, making their practical implementation difficult. These challenges include:

- Extremely costly function evaluations for large datasets or large models.
- High-dimensional and complicated configuration space
- Lack of accessibility to the objective function's gradient and unclear smoothness and convexity of the objective function (Feurer & Hutter, 2019)
- Possibility of the trade-off between two or more objectives such as resource consumption and the model performance (Igel, 2005)
- Dependency of different hyperparameter configurations to various datasets (Feurer & Hutter, 2019).

Previous studies have presented different approaches concerning hyperparameter optimization since the 1990s (King, Feng, & Sutherland, 1995; Kohavi & John, 1995; Michie, Spiegelhalter, & Taylor, 1994). These methods are generally classified into six general groups: search-based algorithms, heuristic algorithms, Bayesian algorithms, multi-fidelity algorithms, population-based algorithms, and reinforcement learning algorithms. Each of these methods has advantages and disadvantages reviewed in the literature review section. In addition to the above methods that provide hyperparameter optimization frameworks, Baydin et al. (2018) developed an approach known as hypergradient descent to optimize one hyperparameter while training the algorithm. During optimization, the hypergradient descent method dynamically updates the learning rate (as the most crucial hyperparameter asserted by Bengio (Bengio, 2012)). Complicated calculations are not required in this method to optimize the learning rate, and only one additional copy of the original gradient should be stored in memory (Baydin et al., 2018); however, it is limited to one hyperparameter.

The current research aims to achieve two objectives. Firstly, it is possible to extend the hypergradient approach for other hyperparameters. In the present work, a new categorization of hyperparameter types is presented, and accordingly, it is attempted to optimize another hyperparameter using the hypergradient method. Secondly, the hypergradient approach is combined with other hyperparameter optimization methods to increase their efficiency. The complexity order of hyperparameter optimization methods (as mentioned in the literature review section) is dependent primarily on the number of hyperparameters. With the combination of the hypergradient method with other hyperparameter optimization approaches, all the hyperparameters are optimized, and the response time is reduced by decreasing the space dimensions. Consequently, the optimization algorithm efficiency is increased.

The rest of the paper is organized as follows. In the second section, the problem statement and its formulation are presented. The hyperparameter optimization algorithms and their advantages and disadvantages of these methods are reviewed in the third section. The fourth section deals with the developed hypergradient HPO approach. The efficiency of the proposed approach is evaluated in Section 5. The sixth section offers the conclusion and recommendations for future works.

2. Statement of the problem

Assume $\mathcal{A}$ represent a learning algorithm with H hyperparameters. Let the h^{th} hyperparameter domain be denoted by Λ_h and the overall hyperparameter configuration space $\Lambda = \Lambda_1 \times \Lambda_2 \times ... \times \Lambda_H$. Moreover, $\boldsymbol{\lambda} \in \Lambda$ denotes a vector of hyperparameters, and $\mathcal{A}_{\boldsymbol{\lambda}}$ shows $\mathcal{A}$ with its hyperparameters instantiated to $\boldsymbol{\lambda}$ (Feurer & Hutter, 2019). Then, the following problem is to be solved in an HPO problem:

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda}\in\Lambda}{\operatorname{argmin}}\, \mathbb{E}_{(\mathrm{D}_{\mathrm{train}},\mathrm{D}_{\mathrm{validation}})}\, V(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, \mathrm{D}_{\mathrm{train}}, \mathrm{D}_{\mathrm{validation}}), \tag{1}$$

where $V(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, \mathrm{D}_{\mathrm{train}}, \mathrm{D}_{\mathrm{validation}})$ shows the loss of a model created by algorithm $\mathcal{A}$ with hyperparameters $\boldsymbol{\lambda}$ on the training data $\mathrm{D}_{\mathrm{train}}$ assessed on the validation data $\mathrm{D}_{\mathrm{validation}}$ (Seifi et al., 2021). The existing works in literature that have proposed different methods to solve this problem are discussed as follows.

3. Review of Related Literature

Studies since the 1990s have presented different methods concerning hyperparameter optimization. It is possible to classify these methods into six general groups according to the core of the used algorithm (Indeed some of category's stem from other one but for their importance they have classified into separate classes. For example, population-based algorithms generally are heuristic methods but here are considered as a different segment). This classification and some of the main algorithms in each class are shown in Fig. 1.

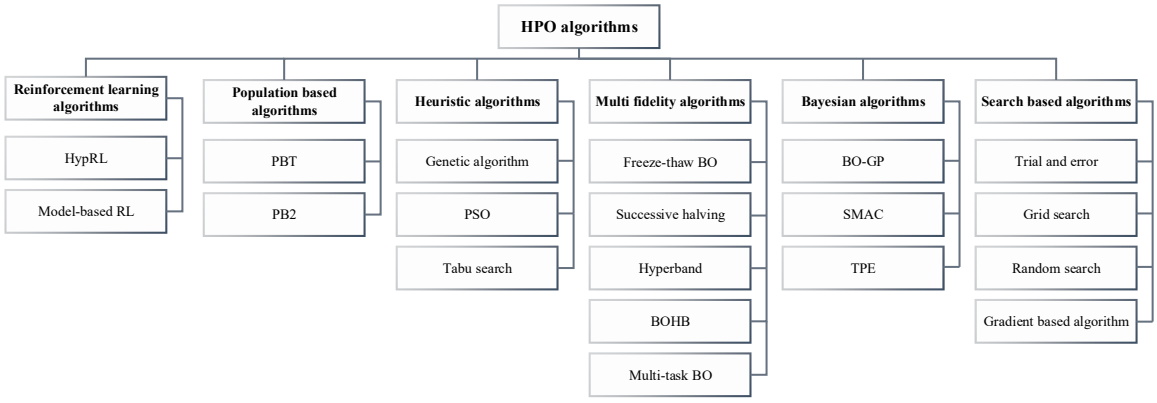

Fig. 1. Classification of HPO algorithms

3.1. HPO algorithms

The first class includes search-based optimization methods, which attempt to discover the optimal values of hyperparameters by dividing the space and evaluating different options for them. Since these are simple algorithms, they are employed in problems with small response space and less-costly evaluation of hyperparameters. However, their efficiency is lost when the space resulting from the hyperparameters is enlarged, or the evaluation cost increases. Trial and error, Grid search, Random search, and Gradient-based algorithms are some the methods of this group which have been used in HPO literature (Bergstra & Bengio, 2012; Chadha & Kaushik, 2022; Hutter, Kotthoff, & Vanschoren, 2019; Sharma et al., 2021). Bayesian optimization is considered an optimization approach to optimize hyperparameters, in which new points are chosen for search based on previous evaluations. Here, firstly, a surrogate model is fitted on the previous data, and then, based on it and based on an acquisition function, a point is selected with the highest desirability. This point is then presented to the main model for training (Injadat, Salo, Nassif, Essex, & Shami, 2018). The main model is trained based on the new hyperparameters, followed by updating the data set. These actions go on as long as the conditions for ending the algorithm are not satisfied (Snoek, Larochelle, & Adams, 2012). Since the surrogate optimization model involves a lower cost than the original model, the Bayesian optimization approach has higher efficiency than the search-based approach since previous information is used in Bayesian optimization. However, it is limited to parallelization (Yang & Shami, 2020). The surrogate model employed in this algorithm principally includes three classes: Gaussian process (Seeger, 2004), random forest (Hutter, Hoos, & Leyton-Brown, 2011), and tree Parzen estimator (Bergstra, Bardenet, Bengio, & Kégl, 2011).

The third class includes muti-fidelity optimization methods. Generally, optimization of hyperparameters has always been a time-consuming process. Nowadays, implementing these processes has become more costly than ever because of the increased complexity of machine learning methods and the increased volume of input data (Feurer & Hutter, 2019). Hence, researchers have always been concerned with reducing the implementation time of hyperparameters optimization. The multi-fidelity optimization method is one of the methods proposed in this regard that attempts to shorten execution time by managing the available budget. In these algorithms, the configurations with poor performance are rejected after each round of hyperparameter evaluation. In contrast, hyperparameter configurations with good performance are evaluated on the entire training set (Yang & Shami, 2020). Execution time is saved by implementing this policy and discarding poor-performing hyperparameters. Multi-fidelity algorithms are classified into two general categories. The first group includes learning curve-based methods. During optimization, these methods forecast the performance of a set of hyperparameters under evaluation if more resources are allocated and stop their evaluation if adding resources does not result in their improvement (Falkner et al., 2018; Feurer & Hutter, 2019). The other group involves methods that exclude some hyperparameters with poorer performance than others at each evaluation step. Hence, the budget required to evaluate the remaining group of members increases (Karnin, Koren, & Somekh, 2013; Li, Jamieson, DeSalvo, Rostamizadeh, & Talwalkar, 2017; Zöller & Huber, 2021).

Table 1
The comparison of standard HPO algorithms

HPO Group	HPO method	Strengths	Weaknesses	Time complexity
Search-based algorithms	Trial and error	1. Simple implementation	1. Needing prior knowledge 2. It cannot be used for the large number of HPs 3. It cannot be for methods with time-consuming evaluation processes 4. It cannot be used for the nonlinear interactions of the HPs	$\mathcal{O}(n)$
	Grid search	1. Simple implementation 2. Paralleling part of the algorithm is possible	1. It is inefficient for high-dimensionality hyperparameter configuration space 2. Continuous HPs are not supported	$\mathcal{O}(n^d)$
	Random search	1. Possibility of parallel executions 2. It is more efficient in using computational budget compared to GS 3. It is a helpful approach to initiate the search process	1. The experience of past evaluations is not used for increasing productivity 2. It is efficient for conditional HPs	$\mathcal{O}(n)$
	Gradient-based algorithm	1. Higher speed of convergence compared to other methods of this group	1. It can get stuck in local optimization if the objective function is not convex 2. It can be applied only to continuous HPs	$\mathcal{O}(n^d)$
Bayesian algorithms	BO-GP	1. The experience of past evaluations is used for increasing productivity 2. Higher speed of convergence in continuous HPs	1. Poor capacity for parallelization 2. Inefficient for conditional HPs	$\mathcal{O}(n^3)$
	BO-RF	1. All HPs types are supported	1. Poor capacity for parallelization	$\mathcal{O}(n \log n)$
	BO-TPE	1. It shows acceptable performance regarding structured HPO tasks 2. All HPs types are supported	1. Poor capacity for parallelization	$\mathcal{O}(n \log n)$
Multi-fidelity algorithms	Freeze-thaw Bayesian optimization	1. The experience of past evaluations is used for increasing productivity 2. Computational budget is better managed, and wastage is avoided	1. Poor capacity for parallelization	$\mathcal{O}(n^3)$
	Successive halving	1. Simple implementation 2. Real-world conditions, such as budget constraints, are considered 3. Possibility of parallelization	1. It is highly dependent on the amount of the budget allocated	$\mathcal{O}(n)$
	Hyperband	1. A balance is achieved for the number of configurations and budget 2. Possibility of parallelization	1. It needs representativeness of subsets with small budgets	$\mathcal{O}(n \log n)$
	BOHB	1. Robust final performance and anytime performance 2. Efficiency in all HPs types 3. Possibility of parallelization	1. It needs representativeness of subsets with small budgets 2. Inefficient for conditional HPs	$\mathcal{O}(n \log n)$
Heuristic algorithms	Genetic algorithm	1. Efficiency in all HPs types 2. Good initialization is not required	1. Poor capacity for parallelization	$\mathcal{O}(n^2)$
	PSO	1. Efficiency in all HPs types 2. Possibility of parallelization	1. Require proper initialization	$\mathcal{O}(n \log n)$
	Tabu search	1. Efficiency in all HPs types 2. Possibility of parallelization	1. Require proper initialization	$\mathcal{O}(n^2)$
Population-based algorithms	PBT	1. It is more efficient in computational resources 2. Hyperparameters and parameters are optimized together	1. It performs poorly in primary steps 2. It performs poorly in continuous hyperparameters	–
	PB2	1. Higher efficiency in computational resources 2. Hyperparameters and parameters are optimized together 3. The search is efficiently guided by using a probabilistic model	1. It performs poorly in primary steps 2. It performs poorly in continuous hyperparameters	–
Reinforcement learning algorithms	HypRL	1. Time complexity is lower than Bayesian 2. Good performance with resources constraints	1. Poor capacity for parallelization 2. Complicated algorithm execution	–
	Model-based RL	1. Accelerate learning by using a predictive model to directly evaluate the performance of hyperparameter configuration. 2. Using the KL divergence between policies to dynamically control the model used avoids the model bias problem.	1. Poor capacity for parallelization 2. Complicated algorithm execution	–

Note. n: Number of instances; d.: Number of dimensions.

The fourth category consists of heuristic algorithms such as genetic algorithm, particle swarm optimization, and Tabu search algorithm, which seek to improve the time for reaching the optimal response by making a balance between exploration and exploitation. Kumar and Haider (2021) and Ibad et al. (2022) have used heuristic algorithms for HPO. The fifth category includes population-based optimization algorithms such as population-based Bandits (Parker-Holder, Nguyen, & Roberts, 2020) and population-based training (Jaderberg et al., 2017), which link between parallel and sequential optimization methods. This method requires less time than the sequential optimization methods, and concurrently, less computational resources are used than the parallel methods. Therefore, this method takes advantage of both groups of methods. The last group contains reinforcement learning algorithms. These studies propose a set of actions to determine the environment where the agent learns the way of moving through the hyperparameters' space and gaining the best solution. These actions identify the hyperparameters needing evaluation. The selection of hyperparameters is made sequentially using a policy (similar to the acquisition function in the Bayesian method) that provides a balance in the trade-off between exploitation and exploration (Jomaa, Grabocka, & Schmidt-Thieme, 2019). HypRL (Jomaa et al., 2019) and model-based reinforcement learning algorithms (Wu, Chen, & Liu, 2020) are some examples of this group.

3.2. Comparison of HPO algorithms

There exist many hyperparameter optimization methods for various applications. Thus, choosing the proper optimization method for each machine learning algorithm is crucial. The problem characteristics and conditions, including the hyperparameter characteristics and their relationships, govern the selection process of the algorithm with the highest efficacy. Additionally, it is necessary to consider general optimization conditions, including constraints in using computational resources, time constraints, the ability to execute more complicated optimization algorithms, and the objective function evaluation cost. Table 1 classifies the type of HBO methods, shows the standard hyperparameter optimization methods, and summarizes their weaknesses, strengths, and the time complexity involved in their implementation. Accordingly, they can be compared to each other, by which the most appropriate method can be chosen.

4. The proposed methodology

With the increased complexity of machine learning methods and the growing data volume involved in them, it takes longer to execute these algorithms and optimize their hyperparameters. Besides, since the online application of these methods is increasing, they should be tuned in a shorter time. However, as observed in Table 1, some hyperparameter optimization methods suffer from the curse of dimensionality. In this case, to obtain a reliable result, the number of evaluations needed often grows with dimensionality. As such, there is a need to develop a method that decreases the size of the search space. Baydin et al. (2018) proposed a method in which the learning rate is dynamically updated during optimization by using the gradient relative to the learning rate of the update rule. This method requires a little additional computation for this "hypergradient" computation. It only requires storing one extra copy of the original gradient in memory and only relies on what is provided by reverse-mode automatic differentiation. Like the structure developed by Baydin et al. (2018), the learning rate is optimized in current research during the model training. Besides, it is possible to increase efficiency if it can be extended to other hyperparameters and combined with other HPO methods. A new classification of hyperparameter types should be provided for generalizing this method to other hyperparameters. DeCastro-García et al. (DeCastro-García et al., 2019) presented a classification for the hyperparameter types as discrete, continuous, conditional, and categorical, based on the nature of hyperparameters. Moreover, the hyperparameter types can be categorized into structural hyperparameters and non-structural hyperparameters. The former directly specify the model's overall structure (e.g., the activation function and layer number in neural networks), and the latter are those that do not require stopping and changing the model if they change during model training (e.g., momentum and learning rate in a neural network). Given this classification, it is possible to generalize the method Baydin et al. (2018) proposed to non-structural hyperparameters. Here, the aim is to extend this method to another hyperparameter termed momentum and then hybridize it with hyperparameter optimization methods. To this end, the regular gradient descent method is considered. Given previous parameters θ_{t-1} and an objective function, the gradient $\nabla f(\theta_{t-1})$ is evaluated. Moving against it so that updated parameters are achieved by

$$\theta_t = \theta_{t-1} - \alpha \nabla f(\theta_{t-1}) - \gamma(\theta_{t-1} - \theta_{t-2}), \tag{2}$$

where γ and α denote the momentum and learning rate, respectively. In addition to this update rule, two update rules are designed for the momentum γ and learning rate α. To obtain the update rule for α, it is required to obtain the partial derivative of the objective function f at the previous time step with respect to the learning rate. In other words, using the chain rule, we have:

$$\frac{\partial f(\theta_{t-1})}{\partial \alpha} = \frac{\partial f(\theta_{t-1})}{\partial \theta_{t-1}} \times \frac{\partial \theta_{t-1}}{\partial \alpha}. \tag{3}$$

Using Eq. (2), θ_{t-1} can be written as:

$$\theta_{t-1} = \theta_{t-2} - \alpha\nabla f(\theta_{t-1}) - \gamma(\theta_{t-2} - \theta_{t-3}). \quad (4)$$

Thus, the $\frac{\partial\theta_{t-1}}{\partial\alpha}$ value in Eq. (3) equals to ($-\nabla f(\theta_{t-2})$) according to Eq. (4). When this value is substituted in Eq. (3), we have:

$$\frac{\partial f(\theta_{t-1})}{\partial\alpha} = \nabla f(\theta_{t-1})\big(-\nabla f(\theta_{t-2})\big). \quad (5)$$

Hence, the update rule for learning rate α is obtained as follow:

$$\alpha_t = \alpha_{t-1} - \beta\frac{\partial f(\theta_{t-1})}{\partial\alpha} = \alpha_{t-1} + \beta\,\nabla f(\theta_{t-1})\nabla f(\theta_{t-2}), \quad (6)$$

where β denotes the learning rate of hypergradient. Similarly, to obtain the update rule for γ, it is required to obtain the partial derivative of the objective f at the previous time step relative to the learning rate. Once again, based on the chain rule, we have:

$$\frac{\partial f(\theta_{t-1})}{\partial\gamma} = \frac{\partial f(\theta_{t-1})}{\partial\theta_{t-1}} \times \frac{\partial\theta_{t-1}}{\partial\gamma}. \quad (7)$$

Based on Eq. (4), the $\frac{\partial\theta_{t-1}}{\partial\gamma}$ value equals to $(\theta_{t-2} - \theta_{t-3})$. When this value is replaced in Eq. (7), we have:

$$\frac{\partial f(\theta_{t-1})}{\partial\gamma} = \nabla f(\theta_{t-1})(-(\theta_{t-2} - \theta_{t-3})). \quad (8)$$

Therefore, the update rule for the learning rate γ is:

$$\gamma_t = \gamma_{t-1} - \eta\frac{\partial f(\theta_{t-1})}{\partial\gamma} = \gamma_{t-1} + \eta\,\nabla f(\theta_{t-1})(\theta_{t-2} - \theta_{t-3}), \quad (9)$$

where η denotes the momentum rate of hypergradient. The α_t and γ_t values are updated by Eq. (6) and Eq. (9) along with the θ_t value in each step. Hence, instead of hyperparameters, they become parameters whose values are specified during the model training. In this situation, the number of hyperparameters that need optimization using an optimization method is decreased. Therefore, the time required for reaching the optimal configuration is reduced. Note here that only the previous step's gradient and the parameters of the two previous steps are required to be stored in memory. Hence, the complexity is not increased in the optimization algorithms. The learning part of the proposed algorithm is shown in Algorithm 1.

Algorithm 1: The Extended hypergradient descent technique

Initialize: $\theta_0, \alpha_0, \gamma_0, \beta, \eta$
  While θ_t not converged do:
    $t \leftarrow t + 1$
    Calculate $\nabla f(\theta_{t-1})$
    $\alpha_t \leftarrow \alpha_{t-1} + \beta\,\nabla f(\theta_{t-1})\nabla f(\theta_{t-2})$
    $\gamma_t \leftarrow \gamma_{t-1} + \eta\,\nabla f(\theta_{t-1})(\theta_{t-2} - \theta_{t-3})$
    $\theta_t \leftarrow \theta_{t-1} - \alpha_t\nabla f(\theta_{t-1}) - \gamma_t(\theta_{t-1} - \theta_{t-2})$
  End while
Return θ_t

5. Performance evaluation

To evaluate the proposed method's performance, we combined this method with various hyperparameter optimization algorithms, and it was tested in two case studies. In the first case study, a convolutional neural network (CNN) is optimized on the CIFAR-10 image database (an object recognition dataset developed by Krizhevsky and Hinton (2009)). The hyperparameters of these networks include two fully connected layer sizes, three convolutional layer sizes, batch size, momentum, and learning rate. The purpose of designing the second case study is to evaluate the method performance in the increased complexity of the prediction problem on the CIFAR-100 dataset. The hyperparameters of the first problem are also considered in the second case, except that they have different network structures (issues such as drop rate, kernel size, the presence of activation function types of each one, and a fixed size convolutional layer in the first case) and the hyperparameter domains are different in two cases. Table 2 presents the domain of the value of these hyperparameters. To tune the hyperparameters, we investigated six HPO algorithms: Hyperband, Async Successive Halving Algorithm (ASHA), Async Hyperband (AHB), Population-Based Bandits (PB2), genetic, and Population-Based Training (PBT). Experiments were run using PyTorch on a machine with Intel(R) Xeon(R) CPU @ 2.30GHz, 13 GB RAM, and 1xTesla K80 having 2496 CUDA cores GPU. The network is optimized using these algorithms with and without the proposed method to illustrate the impact

of the presented framework. All optimization strategies are compared in terms of the best average accuracy evaluation measure. In fact, the best-generated accuracy for the selected hyperparameters of an algorithm is considered at each moment, and the average of this index is calculated for five runs (for each method). In addition to that, the datasets have been divided to train and test segments (70% for training and 30% for test) and the evaluations have been done based on cross validation method. Figures 2 and 3 show the results of these runs.

Table 2
Value domain of hyperparameters

Hyperparameter	Domain (First case study)	Domain (Second case study)
Size of 1st convolutional layer	[4,8,16,32,64,128,256,512]	[8,16,32,64,128]
Size of 2nd convolutional layer	[4,8,16,32,64,128,256,512]	[32,64,128,256,512,1024,2048]
Size of 3rd convolutional layer	[4,8,16,32,64,128,256,512]	[64,128,256,512,1024,2048,4096]
Size of 1st fully connected layer	[128,256,512,1024,2048]	[64,128,256,512,1024,2048,4096]
Size of 2nd fully connected layer	[32,64,128,256]	[32,64,128,256,512,1024,2048]
Batch size	[4,8,16,32,64]	[32,64,128,256]
Momentum	[0.7,0.99]	[0.7,0.99]
Learning rate	[10^{-5},0.2]	[10^{-5},0.2]

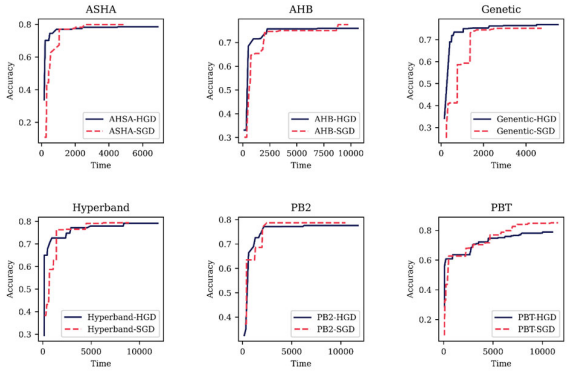

Fig. 2. Evaluation results for six optimization methods on CIFAR-10 dataset with treating momentum and learning rate as two parameters vs. treating them as two hyperparameters. The average validation accuracy over four algorithm evaluations over time (x-axis) is represented by Y-axis. The solid line indicates the algorithm's average accuracy with adjustment of the momentum and learning rates using the hypergradient method. Also, the dashed line indicates the algorithm's average accuracy when giving the momentum and learning rate as two hyperparameters.

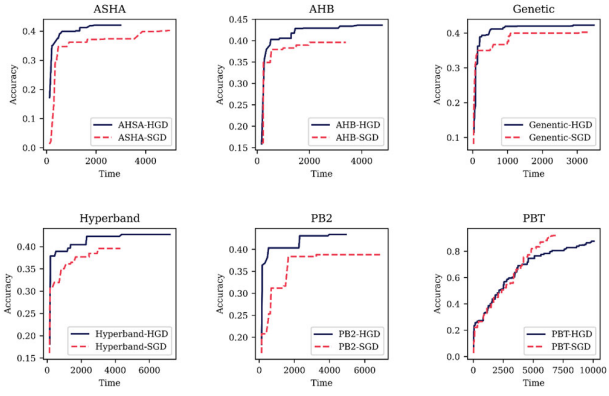

Fig. 3. Evaluation results for six optimization methods on CIFAR-100 dataset with treating momentum and learning rate as two parameters vs. treating them as two hyperparameters. The average validation accuracy over four algorithm evaluations over time (x-axis) is represented by Y-axis. The solid line indicates the algorithm's average accuracy with adjustment of the momentum and learning rates using the hypergradient method. Also, the dashed line indicates the algorithm's average accuracy when giving the momentum and learning rate as two hyperparameters.

In Fig. 2 and Fig. 3, the average accuracy of the different performances of the six algorithms is shown. The solid line in these figures indicates the average accuracy of the algorithm with adjusting the momentum and learning rates using the hypergradient method. The dashed line indicates the average accuracy of the algorithm when two hyperparameters are given to the momentum and learning rate. These diagrams show the significantly better performance of the algorithms in the initial steps when optimizing the momentum and learning rate through the hypergradient method compared to when giving these

two hyperparameters to the algorithm as two optimization variables. As a result of the hypergradient method, the search space is reduced to increase algorithms' efficiency. So, in the primary seconds of execution, if this method is combined with algorithms, more accurate answers are obtained. In addition to these high-quality answers in the first steps, the final produced results by the proposed method are equal to or better than the results without it. So, the proposed method improves performance of the algorithms in both initial steps and final steps.

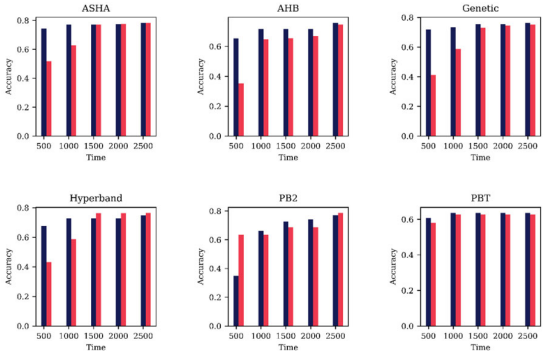

Fig. 4. Comparing the average validation accuracy of the algorithms on the CIFAR-10 dataset at 500, 1000, 1500, 2000, and 2500 seconds of execution. The Y-axis represents the average validation accuracy over four algorithm evaluations over time (x-axis). The blue line indicates the algorithm's average accuracy by adjusting the momentum and learning rates by the hypergradient method. The red line indicates the algorithm's average accuracy when presenting the momentum and learning rate as two hyperparameters.

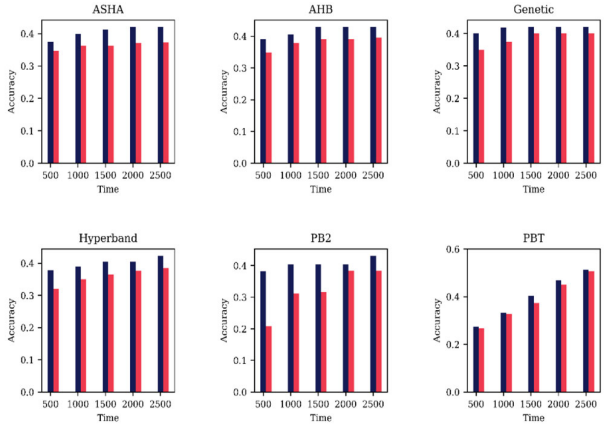

Fig. 5. Comparing the average validation accuracy of the algorithms on the CIFAR-10 dataset at 500, 1000, 1500, 2000, and 2500 seconds of execution. The average validation accuracy over four algorithm evaluations over time (x-axis) is represented by Y-axis. The blue line indicates the algorithm's average accuracy by adjusting the momentum and learning rates by the hypergradient method. The red line indicates the algorithm's average accuracy when presenting the momentum and learning rate as two hyperparameters. To better compare algorithms, the average accuracy after 500s (to show the initial performance of them) and after 3000s (to show the final performance) are compared in Table 3 and Table 4.

Table 3
Comparison of the proposed algorithm results for Cifar10 dataset

		Average accuracy					
Time (s)	**Algorithm**	**ASHA**	**AHB**	**Genetic**	**HyperBand**	**PBT**	**PB2**
500	SGD	0.5164	0.3528	0.4126	0.4317	0.5804	**0.6347**
	HGD	**0.7445**	**0.6526**	**0.7188**	**0.6768**	**0.6073**	0.3491
3000	SGD	**0.7986**	**0.7986**	0.7517	0.7646	0.7033	**0.7869**
	HGD	0.7822	0.7570	**0.7636**	**0.7719**	**0.7038**	0.7714

To have a better comparison of the output improvements in the primary steps, Fig. 4 and Fig. 5 give the average validation accuracy of the algorithms at 500, 1,000, 1,500, 2,000, and 2,500 seconds of execution. The blue line in these figures indicates the algorithm's average accuracy when adjusting the momentum and learning rates using the hypergradient method. The red

line shows the algorithm's average accuracy when presenting the momentum and learning rate as two hyperparameters. Figures 4 and 5 indicate that if we treat the momentum and learning rate as two parameters and take them out of the search space of the HPO algorithm, in 88.3% of cases, the results in the initial steps are better than the situation when considering these two variables as two hyperparameters. As already mentioned, gaining high-quality outcomes in a shorter time is highly beneficial in new hyperparameter optimization applications. Thus, it is expected to apply this method in such problems.

Table 4
Comparison of the proposed algorithm results for Cifar100 dataset

		Average accuracy					
Time (s)	**Algorithm**	**ASHA**	**AHB**	**Genetic**	**HyperBand**	**PBT**	**PB2**
500	SGD	0.3467	0.3493	0.3500	0.3196	0.2669	0.2077
	HGD	**0.3749**	**0.3899**	**0.4010**	**0.3787**	**0.2733**	**0.3817**
3000	SGD	0.3736	0.3960	0.3997	0.3847	0.5473	0.3837
	HGD	**0.4204**	**0.4290**	**0.4226**	**0.4229**	**0.5963**	**0.4034**

Based on Table 3 and Table 4 the performance of the algorithms has been increased 36.62% and 23.16% for Cifar10 and Cifar100 dataset respectively in early stages (500 seconds average accuracy). Additionally, the final performance has been decreased just 1.10% in Cifar10 and has been increased 8.43% for Cifar100 dataset which shows that the proposed method increases the performance in early stage while doesn't affect their final performance significantly in a negative way. Despite improving algorithm performance by this method in the primary steps, its combination with population-based methods does not result in a significant increase in their efficiency. Population-based optimization methods copy the parameters from the answers that have had a good performance in some steps. As such, the conversion of the momentum and the learning rate as two hyperparameters into two parameters using the presented method results in copying both well-performing answers to other answers. Besides, there is a high learning rate in the initial steps in the hypergradient method, and it reduces in the final steps. Copying its final value for the starting values of other points in the solution space decreases the convergence speed in these methods. Hence, the combination of the presented method with the population-based methods does not significantly improve their performance.

6. Conclusion and future work

The popularity of machine learning methods has increased in recent years because of their exceptional performance. Moreover, these methods' performance depends on the exact setting of hyperparameters. Previous studies have presented different methods for optimizing the hyperparameters of machine learning models. However, the common point in these methods is a dependency of the time complexity of these methods on the number of hyperparameters that should be optimized. Thus, the present study aimed to develop a new classification of the hyperparameter types and adjust some deep learning hyperparameters during training by using the hypergradient method. Hence, the search space of the optimization algorithm can be reduced. The proposed method can optimize the hyperparameters during training, with the need for only the parameters of the previous two steps and the gradient of the previous step. Consequently, the complexity order of the optimization model is not increased. Besides, it is possible to combine this method with all HPO methods. As indicated by the case studies, this method caused the search space reduction and has improved the average accuracy 36.62% and 23.16% for Cifar10 and Cifar100 dataset respectively in the primary steps while it has not affected their final performance significantly in a negative way. For future studies, it is recommended to extend this method to other non-structural hyperparameters.

Conflict of Interest Statement

The authors declare that they have no conflict of interest.

Data Availability Statement

The data used in this work is available upon request.

References

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., & Wood, F. (2018). *Online learning rate adaptation with hypergradient descent*. Paper presented at the ICLR 2018 Conference.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade* (pp. 437-478): Springer.

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems, 24*.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of machine learning research, 13*(2).

Chadha, A., & Kaushik, B. (2022). A Hybrid Deep Learning Model Using Grid Search and Cross-Validation for Effective Classification and Prediction of Suicidal Ideation from Social Network Data. *New Generation Computing, 40*(4), 889-914.

DeCastro-García, N., Muñoz Castañeda, Á. L., Escudero García, D., & Carriegos, M. V. (2019). Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm. *Complexity, 2019*.

Falkner, S., Klein, A., & Hutter, F. (2018). *BOHB: Robust and efficient hyperparameter optimization at scale.* Paper presented at the International Conference on Machine Learning.

Feurer, M., & Hutter, F. (2019). Hyperparameter optimization. In *Automated machine learning* (pp. 3-33): Springer, Cham.

Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2011). *Sequential model-based optimization for general algorithm configuration.* Paper presented at the International conference on learning and intelligent optimization.

Hutter, F., Kotthoff, L., & Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*: Springer Nature.

Ibad, T., Abdulkadir, S. J., Aziz, N., Ragab, M. G., & Al-Tashi, Q. (2022). Hyperparameter optimization of evolving spiking neural network for time-series classification. *New Generation Computing, 40*(1), 377-397.

Igel, C. (2005). *Multi-objective model selection for support vector machines.* Paper presented at the International conference on evolutionary multi-criterion optimization.

Injadat, M., Salo, F., Nassif, A. B., Essex, A., & Shami, A. (2018). *Bayesian optimization with machine learning algorithms towards anomaly detection.* Paper presented at the 2018 IEEE global communications conference (GLOBECOM).

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., . . . Simonyan, K. (2017). Population based training of neural networks. *arXiv preprint arXiv:1711.09846*.

Jomaa, H. S., Grabocka, J., & Schmidt-Thieme, L. (2019). Hyp-rl: Hyperparameter optimization by reinforcement learning. *arXiv preprint arXiv:1906.11527*.

Karnin, Z., Koren, T., & Somekh, O. (2013). *Almost optimal exploration in multi-armed bandits.* Paper presented at the International Conference on Machine Learning.

King, R. D., Feng, C., & Sutherland, A. (1995). Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence an International Journal, 9*(3), 289-333.

Kohavi, R., & John, G. H. (1995). Automatic parameter selection by minimizing estimated error. In *Machine Learning Proceedings 1995* (pp. 304-312): Elsevier.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images.

Kumar, K., & Haider, M. T. U. (2021). Enhanced prediction of intra-day stock market using metaheuristic optimization on RNN–LSTM network. *New Generation Computing, 39*, 231-272.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research, 18*(1), 6765-6816.

Liu, X., Wu, J., & Chen, S. (2022). A Context-Based Meta-Reinforcement Learning Approach to Efficient Hyperparameter Optimization. *Neurocomputing*.

Michie, D., Spiegelhalter, D. J., & Taylor, C. C. (1994). Machine learning, neural and statistical classification.

Parker-Holder, J., Nguyen, V., & Roberts, S. (2020). Provably efficient online hyperparameter optimization with population-based bandits. *Advances in neural information processing systems, 33*, 17200-17211.

Seeger, M. (2004). Gaussian processes for machine learning. *International journal of neural systems, 14*(02), 69-106.

Seifi, F., Azizi, M. J., & Niaki, S. T. A. (2021). A data-driven robust optimization algorithm for black-box cases: An application to hyper-parameter optimization of machine learning algorithms. *Computers & Industrial Engineering, 160*, 107581.

Sharma, N., Dev, J., Mangla, M., Wadhwa, V. M., Mohanty, S. N., & Kakkar, D. (2021). A heterogeneous ensemble forecasting model for disease prediction. *New Generation Computing*, 1-15.

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems, 25*.

Wu, J., Chen, S., & Liu, X. (2020). Efficient hyperparameter optimization through model-based reinforcement learning. *Neurocomputing, 409*, 381-393.

Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing, 415*, 295-316.

Yao, C., Cai, D., Bu, J., & Chen, G. (2017). Pre-training the deep generative models with adaptive hyperparameter optimization. *Neurocomputing, 247*, 144-155.

Zöller, M.-A., & Huber, M. F. (2021). Benchmark and survey of automated machine learning frameworks. *Journal of Artificial Intelligence Research, 70*, 409-472.