Les Cahiers du GERAD ISSN: 0711–2440

Use of static surrogates in hyperparameter optimization

D. Lakhmiri, S. Le Digabel

G–2021–10

March 2021

GERAD HEC Montréal
3000, chemin de la Côte-Sainte-Catherine
Montréal (Québec) Canada H3T 2A7

Tél. : 514 340-6053
Téléc. : 514 340-5665
info@gerad.ca
www.gerad.ca

Use of static surrogates in hyperparameter optimization

Dounia Lakhmiri
Sébastien Le Digabel

GERAD & Department of Mathematics and Industrial Engineering, Polytechnique Montréal, Montréal (Québec), Canada

dounia.lakhmiri@polymtl.ca
sebastien.le.digabel@gerad.ca

March 2021
Les Cahiers du GERAD
G–2021–10

Abstract : Optimizing the hyperparameters and architecture of a neural network is a long yet necessary phase in the development of any new application. This consuming process can benefit from the elaboration of strategies designed to quickly discard low quality configurations and focus on more promising candidates. This work aims at enhancing HyperNOMAD, a library that adapts a direct search derivative-free optimization algorithm to tune both the architecture and the training of a neural network simultaneously, by targeting two keys steps of its execution and exploiting cheap approximations in the form of static surrogates to trigger the early stopping of the evaluation of a configuration and the ranking of pools of candidates. These additions to HyperNOMAD are shown to improve on its resources consumption without harming the quality of the proposed solutions.

Keywords: Hyperparameter optimization (HPO), Derivative-free optimization (DFO), Blackbox optimization (BBO)

Acknowledgements: This work is in part supported by the NSERC Alliance grant 544900–19 in collaboration with Huawei-Canada.

1 Introduction

The efficacy of deep neural networks (DNN) to discover patterns in complex datasets is seen throughout multiple applications where the appropriate variant of a DNN often manages to score higher than human experts or other machine learning algorithms and even to push the current state of the art. A particular class of DNN includes convolutional neural networks (CNN) which are used for image classification, image segmentation, or object detection. They have been gaining in popularity and attention from the scientific community in the recent years as they are at heart of important technological advances such as imitation learning [14] or medical imagery analysis [23] to name a few.

An important challenge when adopting a deep learning approach for a new task is to find the appropriate neural network by deciding on a set of hyperparameters that determine the architecture, i.e, the number of layers, type of blocks and connection, and the training regime. The final score of the network is highly sensitive to the tuning of these hyperparameters and there is no rigorous or intuitive rule that directly provides a range for their optimum values. This hyperparameter optimization (HPO) problem can therefore be framed as a blackbox optimization problem where the objective value is the result of a computation with no analytical formula and no known derivatives. Moreover, this process is time consuming and expensive in computational resources as each configuration trial is equivalent to training a new network long enough to infer its generalization score. In a supervised learning setting, the HPO problem can be expressed as

$$\min_{\Phi} f(\Phi, \theta^*) \quad \text{with } \theta^* \in \operatorname*{argmin}_{\theta} \mathcal{L}_{\Phi,\theta}(X, Y) \tag{1}$$

where f is the objective function that corresponds to a measure of performance of the neural network, usually the validation loss, Φ is the mixed integer vector of all the hyperparameters that define the network, X, Y are the validation data and labels, $\mathcal{L}$ is the loss function that the network optimizes during the training by updating the weights θ.

Derivative-free optimization (DFO) [2, 10] provides a class of methods that are well suited to tackle such blackbox HPO problems as they do not need the explicit expression of the objective function and/or the constraints, nor do they rely on the derivatives for their execution. Two classes of DFO algorithms can be defined: Model-based and direct search methods. Model-based algorithms use a static or dynamic surrogate function $\hat{f}$ as an approximation of the true objective f to guide the optimization. Static surrogates, also called simplified physics surrogates, are defined before the start of the optimization and remain unchanged during the execution in contrast to dynamic surrogates, such as Gaussian processes that are updated with each iteration of the DFO method or with each new evaluation of the objective function. Direct search methods base their exploration of the search space solely on the values of the evaluated points and explore said space through a pattern as it is the case for the Mesh Adaptive Direct Search (MADS) [1] or on a simplex in the case of the Nelder-Mead algorithm [24]. The properties of DFO methods explain their popularity in the context of HPO of deep neural networks where they are often included in specialized libraries such as Hyperopt [6] or Oríon [7]. Similarly, the HyperNOMAD toolbox [18, 19] is developed as an adaptation of MADS to simultaneously optimize the architecture and the training phase of a CNN for a given dataset as expressed in (1). This open source library was shown to have a competitive performance against other popular approaches such as Bayesian optimization [4] or a random search [5] on the MNIST [21], Fashion-MNIST [29] and CIFAR-10 [17] datasets.

The objective of this work is to develop a set of protocols that speed up the HPO process in HyperNOMAD. The proposed measures are based on the use of two static surrogates that affect different steps of the algorithm execution: first the training log of the best encountered network serves as a baseline of comparison with the currently evaluated point and allows for early stopping decisions if the performance of the current network seems unsatisfactory. Second, another static surrogate is employed to rank the candidates to be evaluated at each iteration of HyperNOMAD which applies an opportunistic evaluation strategy, meaning that an iteration is interrupted as soon as a better score

is recorded, therefore disregarding the other pool of candidates. The implementation of these two surrogates is shown to significantly speed up the HPO process thus saving expensive resources.

The rest of the document is structured as follows: Section 2 provides an overview of the literature regarding strategies for early stopping and accelerating the costly HPO problem. Section 3 goes through the algorithm behind HyperNOMAD to highlight where the added strategies are implemented. Section 4 dives into the details of the proposed speed ups and the management of each surrogate. Finally, Section 5 compares the original version of HyperNOMAD with the proposed additions and a synthesis of these results is presented in the discussion.

2 Related work

The common problem of fine tuning a neural network to yield the best performance measure for a specific dataset is highly complex and consuming in terms of time and computational resources, so much so that it is often treated in two separate steps: one for finding the neural architecture and one for optimizing the hyperparameters related to the training of the network. Regardless of which phase is considered, the problem can be formulated as an expensive blackbox optimization problem [2, 10] since the evaluation of the objective function is equivalent to training a new neural network with a specific configuration for a certain amount of time in order to estimate its final accuracy. That necessary amount of time is unknown a priori and has a direct impact on the quantity of resources needed to carry out this task. Moreover, experience shows that not every hyperparameter configuration can yield satisfactory results and it is best to develop the tools that quickly detect such cases and prematurely stop their training, thus saving valuable resources than can be allocated to more promising candidates.

Early stopping is first introduced as a regularization technique to ovoid the overfitting issue that arises in machine learning. Theoretically, the training of a network, and more generally any machine learning algorithm, should be interrupted as soon as the validation error starts to increase as that behaviour indicates a training saturation and an overfitting to the training data. However, as the authors of [26] point out, this criteria can not be directly applied on a real validation error curve that tend to be irregular This work also provides a set of adapted criteria that prompt early stopping when the increase of the validation error exceeds a predetermined threshold, when the deterioration of the validation error is greater than the improvement of the training error or when the generalisation error consistently increases over a certain number of *epochs*. An epoch describes one pass on the entire training data. The training usually consists in multiple epochs before reaching convergence. Each of these criteria is shown to provide good a trade-off between the overall training time and the final performance of the network on a collection of problems. Early stopping is also exploited in Hyperband [22] which adapts the random search (RS) [5] with an efficient resource management scheme. This algorithm starts with allocating a fraction of the resources to each new configuration and compares the resulting validation loss, half of the the low performing candidates are stopped and the other half is granted more resources and the process is repeated throughout the execution of Hyperband with only the relative top performers being trained with the full resources. Compared to other HPO algorithms such as Bayesian methods, Hyperband allows for a significant speed up on multiple datasets such as MNIST and CIFAR-10. The same principle of "starting many, stopping early, continuing some" is applied in [11] where the authors obtained the best results when discarding the least performing networks after $20\% - 30\%$ of the total training budget. All aforementioned works rely on the observed validation curves, yet they highlight a different aspect of the early stopping decisions. In [26], stopping the training is only dependent on the scores of the network itself whereas [11, 22] stop networks for their relative scores compared to a pool of candidates.

Besides, early stopping strategies can base their decisions on the estimation of future scores instead of the observed ones only. Survey [13] dedicates a section to some of the methods used to estimate the final validation performance, or to a lesser extent, the expected scores in future iterations. These strategies extrapolate the validation or training curves [12] via a regression method by factoring the

observed scores and, possibly, the architectural and learning hyperparameters of the network [3, 16]. The usefulness of these estimators in speeding-up hyperparameter optimization or neural architecture search is shown in each work but their main challenge is their need to be trained on a substantial amount of data before having reliable predictions. Another score estimation route uses low fidelity surrogates such as training a network with a lower epoch budget [30], on a subset of the dataset [15, 28] or on lower resolution images [8]. The predicted scores with these methods are expected to be under-estimations of the real validation scores, however they can still be used as a ranking tool to separate the candidate configurations between the top and low performers as long as a good balance in the low fidelity estimate is found so that the under-estimations are not too large.

3 The HyperNOMAD package

The open-source library HyperNOMAD [18, 19] is designed as an adaptation of the NOMAD software [20] to optimize the hyperparameters of deep neural networks as formulated in (1). This package allows searching for both the architecture and the convolutional network's training regime for a specific dataset. It contains two main components: the blackbox and the optimizer, as illustrated in Figure 1 and presented in the following section.

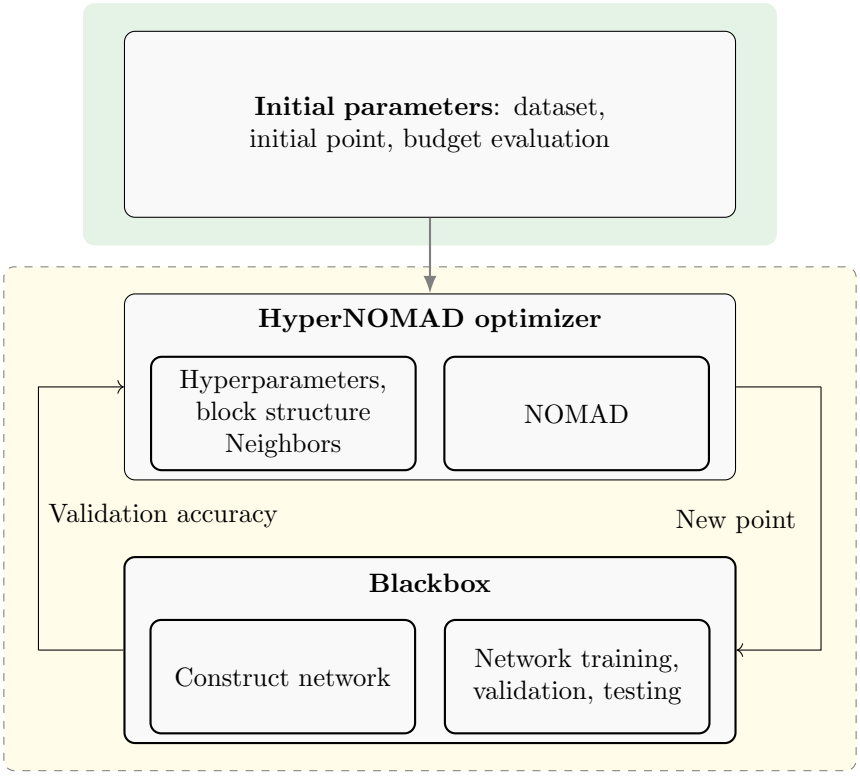


Figure 1: The HyperNOMAD workflow is represented by the communication of its two components. The optimizer suggests new candidates based on the NOMAD software and the blackbox trains the corresponding neural network to return its validation or test accuracy as the objective function value. Image adapted from [19].

3.1 The blackbox

This component regroups the Python and Pytorch [25] modules responsible for creating the equivalent deep neural network to the provided vector of hyperparameters. The blackbox module fully trains the resulting network before returning its validation accuracy or its validation loss as a performance measure. The package provides some computer vision datasets such as MNIST [21], Fashion-MNIST [29], CIFAR-10/100 [17] and STL10 [9] but also allows to plug-in a new dataset if needed.

3.2 The optimizer

The second module is first responsible for handling the collected categorical, integer, or real HPs. Categorical HPs include non-ordinal variables such as the activation function or the choice of the training optimizer and some ordinal variables such as the number of convolutional or fully connected layers. Then, the module launches the optimization through the NOMAD software, which is the official implementation of the Mesh Adaptive Direct Search (MADS) [1] algorithm further discussed in Section 3.3.

In HyperNOMAD, each convolutional layer is defined by five HPs: the number of output channels, the kernel size, the stride, the padding, and the pooling size; And each fully connected layer is determined by the number of nodes it contains. Hence, if n_1 indicates the number of convolutional layers and n_2 the number of fully connected layers, the network's architecture is determined by $5n_1+n_2$ HPs. Considering the remaining HPs such as the optimizer, batch size or dropout rate, the dimension of the complete HPO problem becomes $5n_1 + n_2 + 10$. This dimension varies during the optimization if the values of n_1 and n_2 evolve, explaining why these HPs are considered categorical variables. A change in their values signals a change of search space, which HyperNOMAD handles through the extended polls in Algorithm 1 with a predefined neighborhood structure that states that each CNN has five neighbors. Four of these neighbors are found by adding/subtracting one convolutional/fully connected layer and the fifth neighbor keeps the same architecture and changes the optimizer for the training task.

3.3 The mesh adaptive direct search (MADS) algorithm

As previously stated, the optimizer in HyperNOMAD launches the MADS algorithm to explore the HPs search space. At each iteration k, MADS works on a mesh M_k defined by a mesh size Δ_k that is expanded or reduced depending on whether the previous iteration is successful or not. An iteration is successful if an improvement on the incumbent is recorded; otherwise, it is a failure.

Each iteration is composed of two steps. First, the *search* step is an optional and flexible phase containing any exploration strategy as long as a finite number of trials projected on the mesh M_k are evaluated. Then the *poll* step starts by listing a set of poll points around the incumbent $P_k = \{x_k + \Delta_k d \mid d \in D_k\}$ where D_k is a set of search directions that form a positive basis. The search directions are generated so that the equivalent unit directions become asymptotically dense in the unit sphere. The points $p_k \in P_k$ are evaluated with an *opportunistic* strategy, which means the poll step is interrupted as soon as a configuration scores better than the incumbent x_k, even if other poll points are still available and hence will not be tested. Algorithm 1 summarizes the key steps of the MADS algorithm with an emphasis on where surrogate functions can be integrated.

In the general framework, surrogate functions are optionally plugged into the MADS execution at different stages. For example, a static or dynamic surrogate can be used to explore a wide range of the space during the search phase to suggest new candidates. The poll step can also benefit from surrogate assistance in few distinct aspects, such as deciding on the evaluations to interrupt or providing a ranking so that MADS evaluates the most promising candidate first. Such rankings coupled with the opportunistic strategy are essential to target better solutions faster [27], especially when time and resource constraints are as severe as in the current context. The present work focuses mainly on improving the poll step by targeting both the resource allocation and the ranking aspects of poll points evaluation. The incorporated improvements are discussed in detail in Section 4.

4 Proposed approach

This section details the proposed enhancements to speed-up the resolution of the HPO problem with HyperNOMAD. This objective is achieved by introducing an early stopping mechanism that quickly interrupts the poor performing candidates, coupled with a ranking strategy that allows to evaluate

Algorithm 1: MADS with static surrogates for ranking and early stopping.

[0] Initialization
iteration $k = 0$, configuration x_0, mesh size Δ_0 and mesh M_0,
ranking surrogate S_1 and early stopping surrogate S_2
[1] Search of iteration k (optional)
Construct a set of mesh points and evaluate them
If there is a success, go to **[3]**
[2] Poll of iteration k
Define the poll set P_k of new candidates around x_k,
along search directions that define a positive basis.
Sort the poll points with surrogate S_1
Evaluate the points in P_k with possible early stopping (S_2)
If there is a success, interrupt the evaluations and go to **[3]**
[3] Updates of iteration k
Update Δ_k, x_k, M_k depending on the success of the previous phases
If no stopping condition is satisfied: $k \leftarrow k + 1$ and go to **[1]**

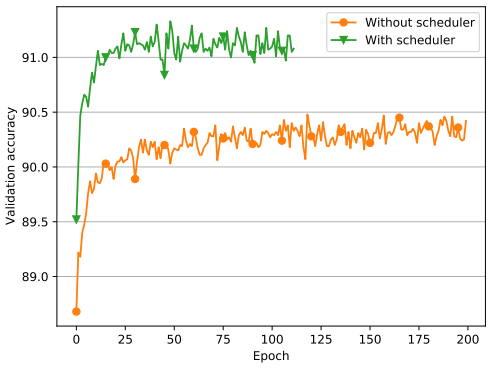
the most promising poll candidates first. All tests in this section are on the MNIST dataset and start the HPO process from two configurations. First with the default initialization of HyperNOMAD, noted p_1, that corresponds to a network with one convolutional layer and two fully connected layers, which amounts to 17 hyperparameters. The second initialization, noted p_2, adds a convolutional layer to the default point, which amounts to 22 hyperparameters.

4.1 Early stopping

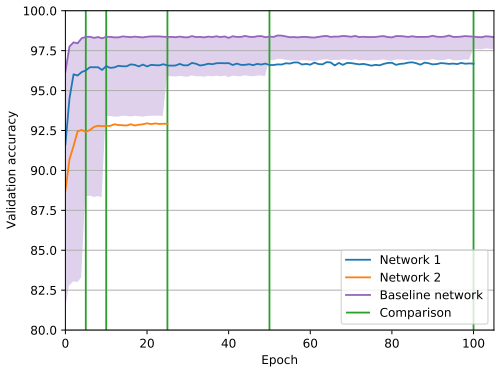
Implementing an adequate early stopping strategy is paramount to the efficient execution of any hyperparameter optimization technique as the complete evaluation of each encountered candidate is merely unreasonable due to the number of wasted resources this direct approach can cause. The challenge is to detect when a network is worth training further and when it needs to be interrupted based on the observed validation curve and compared to other candidates previously trained.

The validation accuracy of a network should gradually improve during the training of a DNN until hitting a plateau or a maximum value after which the accuracy decreases. Such scenarios need to be detected to avoid depleting the remaining training budget on a configuration that can not improve its current validation score any further. However, a plateau does not necessarily mean that the best validation accuracy is reached but rather that the current training HPs, especially the learning rate, need to be updated. To this effect, the PyTorch library [25] provides multiple scheduler variants to manage the learning rate during the training. In particular, the `ReduceLROnPlateau` scheduler can detect plateaus and prompt the desired change in the learning rate. A common strategy is to reduce the learning rate progressively until it reaches a predefined minimum value, which triggers an early stopping. Figure 2a illustrates the benefit of such a mechanism as adding the scheduler allows saving half the training budget in that particular case.

Early stopping is also a decision based on the relative score of the current network compared to the candidates previously evaluated. Such comparison targets the networks whose validation scores are too far from the best solution seen thus far, even if that score gradually improves as the training advances. When training a new candidate, its validation accuracy curve is compared against the best scoring network, called the baseline, at specific epochs: $[5, 10, 25, 50, 100, 125, 150]$ with each milestone having an associated error margin: $[0.5, 0.6, 0.7, 0.8, 0.85, 0.9, 0.95]$. After 5 epochs, a new configuration needs to score at least 50% of the baseline score, 60% at 10 epochs, and so on. As such, the error margins define an envelope under the baseline curve, and any network that scores lower than the allowed error margin is interrupted at the next milestone epoch. This early stopping mechanism is illustrated in Figure 2b. If a better solution is found, the corresponding network becomes the new baseline, and the envelope is redefined with its validation accuracy curve. Consequently, as the HPO advances, better baselines are found, and only the high scoring networks are allowed a high training budget and the low performances are quickly detected, and interrupted.



(a) Example of early stopping due to the scheduler.



(b) Example of early stopping due to the comparison with the baseline network.

Figure 2: Two early stopping criteria: a scheduler that detects a plateau and reduces the learning rate until it hits the lower limits (left) and the comparison with a baseline network that stops any evaluation outside the envelope defined by the relative errors compared to the baseline scores(right).

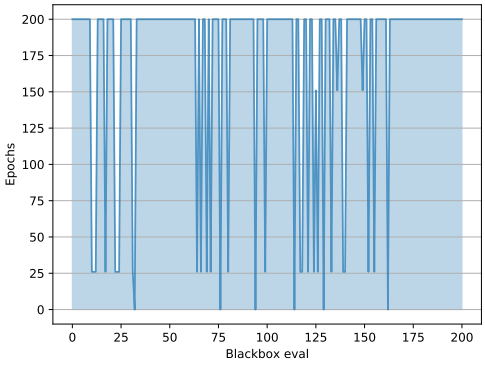
Early stopping effects are first tested on the MNIST dataset by comparing the original version of HyperNOMAD with a variant that implements one early stopping strategy. Every HPO execution starts from the same initial configuration and allows for 200 blackbox evaluations (BBE), meaning that 200 different configurations are trained and tested. The comparison aims at observing the effect on the score of the best solution obtained and the quantity of resources needed to carry out the entire optimization task. Resource consumption is measured in terms of overall wall-clock time and total number of epochs used in training every visited configuration. In this phase, every execution is run on one Nvidia P100 GPU, and the batch size is fixed to 256. The tested early stopping strategies are:

- Default settings of HyperNOMAD: stops if the validation accuracy does not surpass 12% after 25 epochs or when the standard deviation of the validation loss over the last 50 epochs is lower than 10^{-3}.
- Last success: stops the training when the last improvement on the validation accuracy is recorded more than 25 epochs ago.
- Scheduler alone: corresponds to the `ReduceLROnPlateau` scheduler that divides by 10 the learning rate if the validation accuracy has not improve for the last 25 epochs until the learning rate becomes lower than 10^{-8}.
- Baseline and scheduler: takes the previous scheduler scheme and adds the relative scores comparison by defining a baseline and the corresponding envelop as previously detailed.

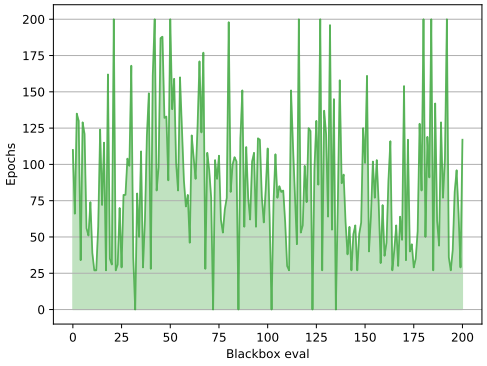
Figures 3 displays the number of epochs used in training each of the 200 configurations evaluated during each execution. The original version is the most resource-consuming, with a mean number of epochs per training at 170. All three added early stopping strategies manage to reduce this measure. The mean number of epochs per training drops to 101.4 with the scheduler, 72.5 with the last success criteria, and combining the scheduler with the baseline comparison brings it down to 45.9. Tables 1a and 1b corroborate this conclusion as the proposed approach is the quickest by a factor of 4 in the second example, plus the score of the best solution does not appear to significantly deteriorate compared to the original version of HyperNOMAD.

4.2 Ranking

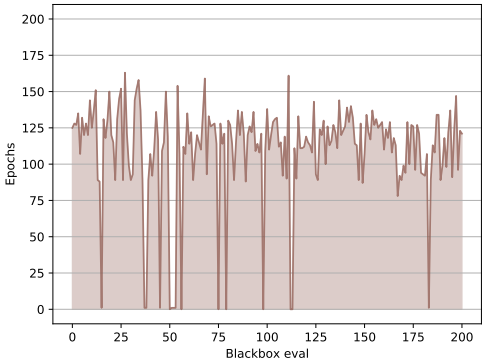
In addition to early stopping, ranking a new pool of candidates based on the expected performance can accelerate the overall HPO process. In HyperNOMAD, the evaluations are opportunistic, meaning that the poll step at iteration k stops as soon as a point $p_k \in P_k$ scores better than the incumbent x_k.



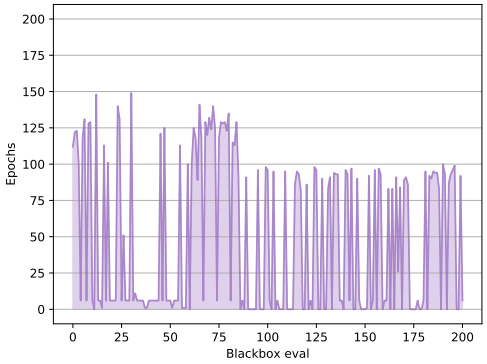
(a) HyperNOMAD**: Stop if the standard deviation of the validation loss is lower than** $1e^{-3}$**.**



(b) HyperNOMAD **+ last success: Stop if the last improvement on the validation score is recorded more than 25 epochs ago.**



(c) HyperNOMAD **+ scheduler: Stop if a plateau is detected with the scheduler.**



(d) HyperNOMAD **+ baseline + scheduler: Stop if a plateau is detected with the scheduler, or if the network scores poorly compared to the baseline.**

Figure 3: Comparison between the resource consumption of HyperNOMAD **and one variant that adds an early stopping strategy, in terms of number of epochs used to train each visited candidate during a single HPO. The HPO starts from the default point p_1 on the MNIST dataset.**

Table 1: Comparison between four early stopping strategies implemented in HyperNOMAD **in terms of top-1 validation accuracy, overall wall clock time and total number of epochs. The variants are launched on MNIST from two starting configurations, are allowed a total of 200 configuration trials and each network can train during a maximum of 200 epochs.**

(a) Scores of HyperNOMAD **on MNIST with each stopping criteria starting from the default point p_1.**

Early stopping strategy	Top-1 val. acc.	Wall-clock time (s)	Total epochs
HyperNOMAD	99.38%	305856	35503
Last success	99.40%	129600	17534
Scheduler	99.29%	170208	21987
Scheduler and baseline	**99.41%**	**126144**	**9681**

(b) Scores of HyperNOMAD **on MNIST with each stopping criteria starting from the configuration p_2.**

Early stopping strategy	Top-1 val. acc.	Wall-clock time (s)	Total epochs
HyperNOMAD	99.43%	280800	32712
Last success	**99.51%**	280800	11480
Scheduler	99.29%	170208	18580
Scheduler and baseline	99.44%	**64800**	**8655**

Therefore, the opportunistic strategy prioritizes fast improvements and, when coupled with an adequate ranking tool, has the potential to explore the search space more efficiently and find the best configurations that much quicker.

Static surrogates, or low fidelity estimates, are approximating functions that do not change during the HPO execution, contrary to dynamic surrogates, which are updated to account for the collected

information with each iteration. Creating a low fidelity surrogate to estimate the accuracy of a DNN can mean training on a fraction of the full epoch budget or on a subset of the dataset in this particular setting. The ideal static surrogate combines a cheap evaluation with a reliable estimation to correctly rank a pool of candidates. Note that in the context of HyperNOMAD and MADS, a good static surrogate does not need to produce accurate approximations of the true objective but rather preserves the ranking order between the candidates.

The effects of ranking the poll points are observed on the MNIST dataset by comparing four static surrogates added to HyperNOMAD. Two surrogates train the candidates on a low training budget, and two others train on a subset of the dataset. Recall that the proposed early stopping strategy results in a mean epoch budget per blackbox evaluation at around 45. Therefore, a static surrogate must use a lower training budget to qualify as a cheap approximation of the true objective evaluation. Similarly, only a small portion of the dataset can be used for the second set of surrogates. The tested surrogates are summarized in Table 2.

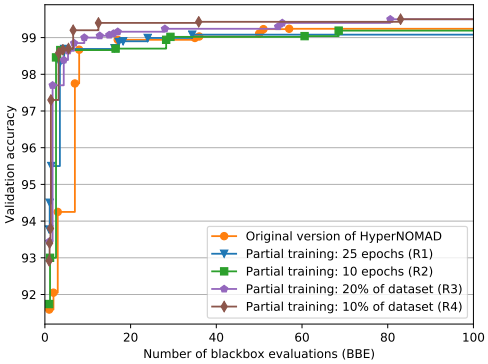
All variants start from the same configurations p_1 and p_2, are allowed 100 blackbox evaluations with a budget of 200 epochs each, and no early stopping criteria are considered at this stage. Evaluating a surrogate must also account in the number of blackbox evaluations as they add to the resource consumption of such HPO optimization. The cost of each surrogate in terms of blackbox evaluation is provided in Table 2. Figures 4 and 5 summarize the results regarding the best validation accuracy per number of blackbox evaluations and overall clock-time. Once again, all executions are run on a single Nvidia P100 GPU.

Table 2: List of the static surrogates tested for ranking the pool candidates. Their evaluation cost in terms of epochs and portion of datasets is compared to a full blackbox evaluation (BBE).

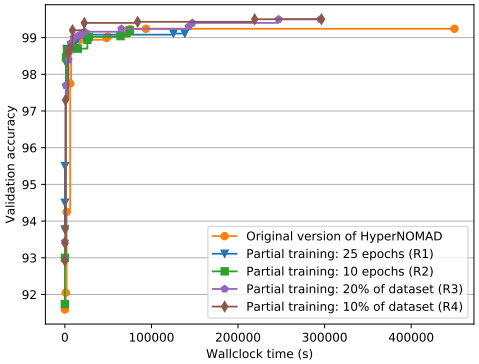
Function	Training budget (epochs)	Portion of dataset	Cost ratio to full BBE
Objective function	200	100%	100%
Surrogate R_1	25	100%	12.5%
Surrogate R_2	10	100%	5%
Surrogate R_3	200	20%	20%
Surrogate R_4	200	10%	10%

Figures 4a and 5a show the effect of adding ranking surrogate on the overall convergence of HyperNOMAD. In Figure 4a, all variants are relatively equivalent and surpass HyperNOMAD in the early stages of the execution. This tendency shifts however after around 10 blackbox evaluations when R_3 and R_4 score better quality solutions faster than the other variants. In fact, R_1 and R_2 have limited benefits to HyperNOMAD since the latter appears to have an equivalent performance in this case. This observation is more obvious in Figure 5a where adding R_1 or R_2 significantly slow down the execution of HyperNOMAD.

Figures 4b and 5b compare the convergence of each algorithm in term of overall wall-clock time. The benefit of adding a sorting surrogate is apparent in both executions as HyperNOMAD is the variant that takes the longest to evaluate the 100 blackbox trials. This is a coherent observation with the fact that every one of the 100 evaluations is equivalent to fully training a network on the entire dataset. Also, in both series of tests, surrogates R_1 and R_2 are the fastest and often terminate the 100 blackbox evaluation budget before all the other variants. It appears that training on such low epoch budgets, even on the full dataset, is faster than training on 10% of the dataset during 200 epochs. However, this observation is expected to change when early stopping is integrated again, which stops R_3 and R_4 from using the full 200 epochs in their evaluation. Plus, since R_1 and R_2 harm the best configuration score, both these strategies are discarded, and it is R_4 that is retained.

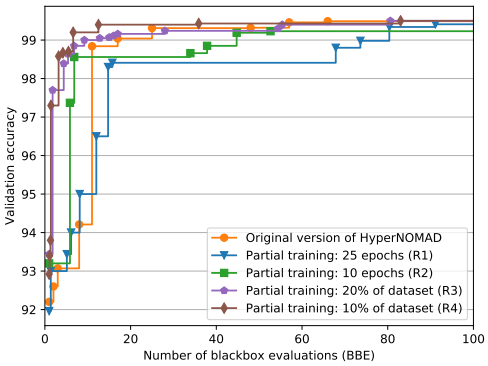


(a) Convergence of each variant in terms of validation accuracy per number of BBE.

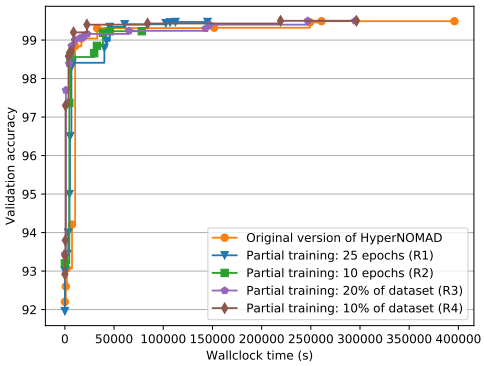


(b) Convergence of each variant in terms of validation accuracy per overall execution time.

Figure 4: Comparison between the original HyperNOMAD **[18, 19] and four variants that implement a strategy to sort the new candidates in order to evaluate the most promising first. Strategies R_1 and R_2 train on a small epoch budget while strategies R_3 and R_4 train on a subset of the data. The optimization is launched on the MNIST dataset from the default point p_1.**



(a) Convergence of each variant in terms of validation accuracy per number of BBE.



(b) Convergence of each variant in terms of validation accuracy per overall execution time.

Figure 5: Comparison between the original HyperNOMAD **[18, 19] and four variants that implement a strategy to sort the new candidates in order to evaluate the most promising first. Strategies R_1 and R_2 train on a small epoch budget while strategies R_3 and R_4 train on a subset of the data. The optimization is launched on the MNIST dataset from the default point p_2.**

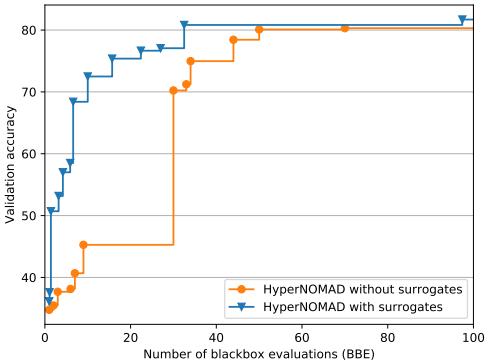
5 Testing

This section incorporates both aspects of Section 4 into the HyperNOMAD framework. Early stopping is triggered from a scheduler or by comparison with the baseline network; and ranking a new pool of candidates is achieved through training each candidate on 10% of the dataset.

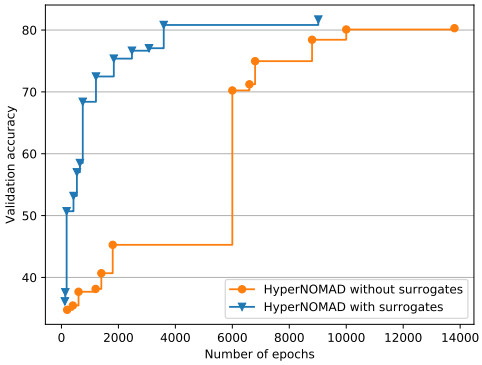
The tests are now launched on the CIFAR-10 dataset, are allowed 100 blackbox evaluations with a maximum training budget of 200 epochs per configuration. This time, each execution is launched on two Nvidia P100 GPUs. One series of tests starts from the default configuration of HyperNOMAD noted p_1, and the second starts from a network with a 5 convolutional layers and 1 fully connected layer, noted p_3, which is equivalent to 36 hyperparameters.

Figure 6 summarizes the results from the comparison between the original version of HyperNOMAD and the one with the proposed enhancements when launched from the default settings p_1. The benefits

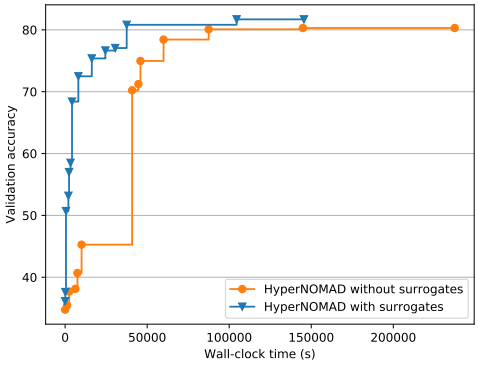
of early stopping and ranking the new candidates are apparent, with the new version scoring better than HyperNOMAD in terms of best validation score per blackbox evaluation budget as seen in Figure 6a. Figure 6b also shows that adding surrogates saves training resources in terms of total number of epochs to get to better quality solution, and Figure 6c corroborates this tendency when comparing the overall wall-clock time to deplete all 100 blackbox evaluations. The second test starts from configuration p_3. Figure 7a shows that the version with the surrogates is slightly ahead when comparing the best validation score per blackbox evaluation budget. Figure 7b illustrates a gain of 27% in total epoch budget, yet this gain is not translated in terms of overall wall-clock time as shown in Figure 7c. The difference is believed to be caused by the communication between the two GPUs used at once.



(a) Convergence in terms of validation accuracy per number of BBE.



(b) Convergence in terms of validation accuracy per number of epochs.

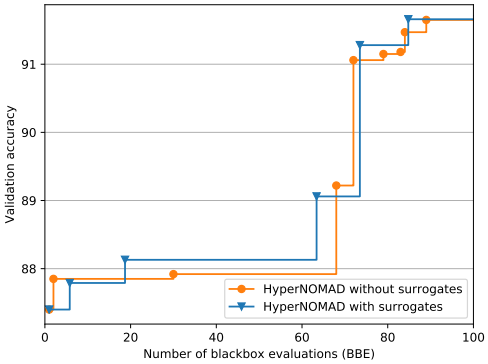


(c) Convergence in terms of validation accuracy per execution time.

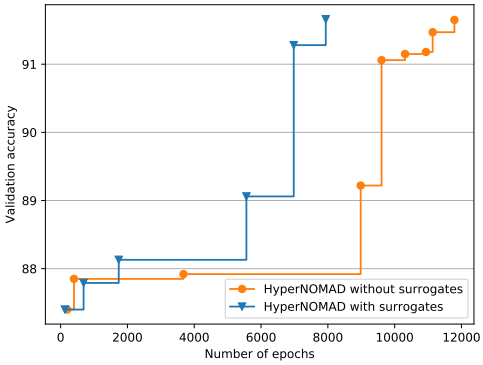
Figure 6: Comparison between HyperNOMAD **with and without surrogates on the CIFAR-10 dataset, starting from the default settings p_1.**

6 Discussion

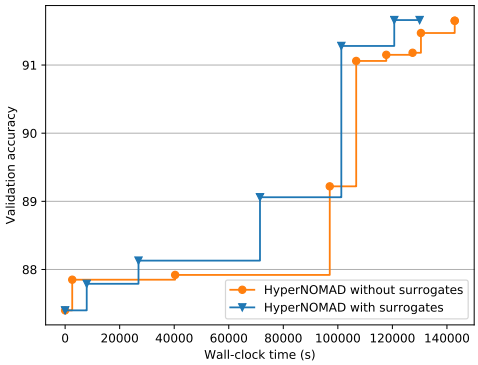
This work proposes a solution to speed up the time and resource-consuming process of optimizing the hyperparameters of deep neural networks, based on incorporating two strategies based on static surrogates into the HyperNOMAD framework. The first defines a new early stopping strategy that quickly and effectively interrupts poorly performing networks. The second allows ranking a pool of new candidates to detect and then evaluate the most promising first. Both techniques are shown individually and collectively to improve on the quality of solution and on the amount of computational resources needed.



(a) Convergence in terms of validation accuracy per number of BBE.



(b) Convergence in terms of validation accuracy per number of epochs.



(c) Convergence in terms of validation accuracy per execution time.

Figure 7: Comparison between HyperNOMAD **with and without surrogates on the CIFAR-10 dataset, starting from the default settings p_3.**

References

[1] C. Audet and J.E. Dennis, Jr. Mesh Adaptive Direct Search Algorithms for Constrained Optimization. SIAM Journal on Optimization, 17(1):188–217, 2006.

[2] C. Audet and W. Hare. Derivative-Free and Blackbox Optimization. Springer Series in Operations Research and Financial Engineering. Springer International Publishing, Cham, Switzerland, 2017.

[3] B. Bowen Baker, G. Otkrist, R. Ramesh, and N. Nikhil. Accelerating neural architecture search using performance prediction. In NIPS Workshop on Meta-Learning, 2017.

[4] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In Advances in neural information processing systems, pages 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.

[5] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13:281–305, 2012.

[6] J. Bergstra, D. Yamins, and D.D. Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In Proceedings of the 30th International Conference on International Conference on Machine Learning, volume 28 of ICML'13, pages I–115–I–123, Atlanta, GA, USA, 2013. JMLR.org.

[7] X. Bouthillier, C. Tsirigotis, F. Corneau-Tremblay, P. Delaunay, R. Askari, D. Suhubdy, M. Noukhovitch, D. Serdyuk, A. Bergeron, P. Henderson, P. Lamblin, M. Bronzi, and C. Beckham. Oríon - Asynchronous Distributed Hyperparameter Optimization. https://github.com/Epistimio/orion, October 2019. (last accessed on 2020-09-19).

[8] P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. arXiv preprint arXiv:1707.08819, 2017.

[9] A. Coates, A. Ng, and H. Lee. An analysis of single-layer networks in unsupervised feature learning. In Proceedings of the fourteenth international conference on artificial intelligence and statistics, pages 215–223, 2011.

[10] A.R. Conn, K. Scheinberg, and L.N. Vicente. Introduction to Derivative-Free Optimization. MOS-SIAM Series on Optimization. SIAM, Philadelphia, 2009.

[11] J. Dodge, G. Ilharco, R. Schwartz, A. Farhadi, H. Hajishirzi, and N. Smith. Fine-tuning pretrained language models: Weight initializations, data orders, and early stopping. arXiv preprint arXiv:2002.06305, 2020.

[12] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In Twenty-Fourth International Joint Conference on Artificial Intelligence, 2015.

[13] T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. arXiv preprint arXiv:1808.05377, 2018.

[14] M. Faria, P. Prado, R. M. S. Julia, and T. Lídia Bononi Paiva. Improving fifa player agents decision-making architectures based on convolutional neural networks through evolutionary techniques. In Ricardo Cerri and Ronaldo C. Prati, editors, Intelligent Systems, pages 371–386, Cham, 2020. Springer International Publishing.

[15] A. Klein, S. Falkner, S. Bartels, P. Hennig, and Frank Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets, 2017.

[16] A. Klein, S. Falkner, T.S. Jost, and F. Hutter. Learning curve prediction with bayesian neural networks. In International Conference on Learning Representations, 2017.

[17] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[18] D. Lakhmiri. HyperNOMAD. https://github.com/bbopt/HyperNOMAD, 2019.

[19] D. Lakhmiri, S. Le Digabel, and C. Tribes. HyperNOMAD: Hyperparameter optimization of deep neural networks using mesh adaptive direct search. Technical Report G-2019-46, Les cahiers du GERAD, 2021. To appear in ACM Transactions on Mathematical Software.

[20] S. Le Digabel. Algorithm 909: NOMAD: Nonlinear Optimization with the MADS algorithm. ACM Transactions on Mathematical Software, 37(4):44:1–44:15, 2011.

[21] Y. LeCun and C. Cortes. MNIST handwritten digit database. http://yann.lecun.com/exdb/mnist, 2010.

[22] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research, 18:1–52, 2018.

[23] G. Marques, D. Agarwal, and I. de la Torre Díez. Automated medical diagnosis of covid-19 through efficientnet convolutional neural network. Applied Software Computing, 96:106691, 2020.

[24] J.A. Nelder and R. Mead. A simplex method for function minimization. The Computer Journal, 7(4):308–313, 1965.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems 32, pages 8024–8035. Curran Associates, Inc., NY, USA, 2019.

[26] L. Prechelt. Early Stopping — But When?, pages 53–67. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[27] L.A. Sarrazin-Mc Cann. Opportunisme et ordonnancement en optimisation sans dérivées. Master's thesis, Polytechnique Montréal, https://publications.polymtl.ca/3099/, 2018.

[28] I. Trofimov, N. Klyuchnikov, M. Salnikov, A. Filippov, and E. Burnaev. Multi-fidelity neural architecture search with knowledge distillation. arXiv preprint arXiv:2006.08341, 2020.

[29] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv preprint arXiv:1708.07747, 2017.

[30] A. Zela, A. Klein, S. Falkner, and F. Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. arXiv preprint arXiv:1807.06906, 2018.