Meta-learning of Black-box Solvers Using Deep Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Black-box optimization does not require any specification on the function to optimize. As such, it represents one of the most general problems in optimization, and is central in many areas such as hyper-parameter tuning. However in many practical cases, one must solve a sequence of black-box problems from functions sampled from a specific class and hence sharing similar patterns. Classical algorithms such as evolutionary or random methods would treat each problem independently and would be oblivious of the general underlying structure. In this paper, we introduce MELBA (MEta bLack Box optimizAtion), an algorithm that exploits the similarities among a given class of functions to learn a task-specific solver that is tailored to efficiently optimize every function from this task. More precisely, given a class of functions, the proposed algorithm learns a Transformer-based Reinforcement Learning (RL) black-box solver. First, the Transformer embeds a previously gathered set of evaluation points and their image through the function into a latent state. Then, the next evaluation point is sampled depending on the latent state. The black-box solver is trained using PPO and the global regret on a training set. We show experimentally the effectiveness of our solvers on various synthetic and real-life tasks including the hyper-parameter optimization of machine learning models (SVM, XGBoost) and demonstrate that our approach is competitive with existing methods.

1 Introduction

Over the past decades, research on black-box optimization has focused on designing algorithms agnostic to the type of problems they can solve. Indeed, the idea behind this approach was to propose algorithms that could solve a large number of optimization problems as different as possible. However, in many real-world applications such as automated machine learning (Hutter et al., 2019) or asset and energy management (Waring et al., 2020; Salimans et al., 2017; Alarie et al., 2021), it is often the case that a similar optimization problem is solved again and again on a regular basis, maintaining the same problem structure but differing in the input data. Moreover, as it has long been known by the celebrated No Free Lunch Theorems for Optimization (Wolpert and Macready, 1997), averaged over all problems, the cost of finding a solution is the same for any optimization algorithm. Thus, whenever an algorithm is efficient on vast classes of problems, it is necessarily suboptimal on a more specific class of functions. Following these observations, we propose in this paper to directly learn black-box optimization algorithms which are tailored to optimize a specific class of functions. To do so, it is necessary to (1) define a class of algorithms that can be learned and act as black-box solvers and (2) define a way to learn these algorithms, despite the fact that the gradients of the functions we wish to optimize are unavailable. To overcome these challenges, we first propose to learn an expressive latent representation of the optimization problem at stake using Transformer-based architectures. Secondly, we show how to train this model by casting the problem

Submitted to the 6th Workshop on Meta-Learning at NeurIPS 2022, New Orleans. Do not distribute.

as a Reinforcement Learnig instance to overcome the problem of non-diffentiability. Using both these ideas, we propose a meta-algorithm called MEta bLack Box optimizAtion (MELBA) which aims at optimizing task-specific black-box functions through meta-learning.

2 Meta-learning for black-box optimization: problem formulation

We consider the problem where we wish to find the maximum of a black-box function $f : \mathcal{X} \to \mathbb{R}$ defined on some input space $\mathcal{X} \subset \mathbb{R}^d$, which is assumed to be sampled from a known class function distribution $\mathcal{F}$ of potentially non-convex and non-differentiable functions. Here, the only thing we assume is that for any black-box function $f \sim \mathcal{F}$ from the distribution, we can query its values at any point of the input domain $\mathcal{X}$. This setting corresponds, for example, to the case where we wish to design a specific algorithm that can optimize the hyper-parameters of an SVM trained on a dataset: choosing a dataset would correspond to sampling a function $f \sim \mathcal{F}$, and training the SVM on some arbitrary hyper-parameters would correspond to querying the value of the function at some point. As an example, Figure 1 displays the loss function associated to the cross-validation error of the two parameters of a SVM on three different datasets, that display strong similarities. To solve this type of problems, most standard black-box optimization (BBO) algorithms rely on a sequential procedure, denoted here by A, that starts with no gathered observations on the function ($o_0 = \{\}$), and iterates as follow: (1) select a candidate solution x_{n+1} that depends on the information gathered so far o_n; , (2) evaluate the objective function (e.g. the corss validation error) at the candidate solution $f(x_{n+1})$; and (3) update information gathered so far $o_{n+1} \leftarrow o_n \cup \{(x_{n+1}, f(x_{n+1}))\}$, $n \leftarrow n + 1$. In practice, the performance of an algorithm A on a black-box function f with a budget of N iterations is generally measured through the difference between the true value of the optimum and the maximum found so far:

$$r(A, f, N) := f^* - \max_{i=1\dots N} f(x_i)\,, \tag{1}$$

where $f^* = \max_{x \in \mathcal{X}} f(x)$ and $x_1, \dots, x_N$ denotes the series of evaluation points chosen by the algorithm A. Thus, given the *a priori* knowledge that the objective is sampled from a distribution $\mathcal{F}$, we can measure the meta-loss of the algorithm A on the class function distribution $\mathcal{F}$ with a budget of N evaluations as follows:

$$R(A, \mathcal{F}, N) = \int_{f \sim \mathcal{F}} r(A, f, N) \mathrm{d}f. \tag{2}$$

In this paper, we investigate the problem of finding the best algorithm A^* that minimize the meta-error for a class of problems $\mathcal{F}$ by minimizing the meta-loss $R(A, \mathcal{F}, N)$. Since it is assumed that we have access to the distribution $\mathcal{F}$, a natural approach would be to minimize its empirical counterpart:

$$A^*(\mathcal{F}, N) \in \operatorname*{arg\,min}_{A \in \mathrm{Alg}} \frac{1}{M} \sum_{j=1}^{M} r(A, f_j, N)\,, \tag{3}$$

where $f_1, \dots, f_M$ are independent problems sampled from the distribution $\mathcal{F}$ and Alg denotes the set of all sequential black-box algorithms. However, this approach suffers from two major drawbacks: (1) how do we define a class of trainable parametrized algorithms Alg large enough to effectively learn efficient sequential algorithms? and (2) how do we perform the optimization over (3) when the quantities $r(A, f_j, N)$ depends on black-box functions f_j for which the gradient is not available?

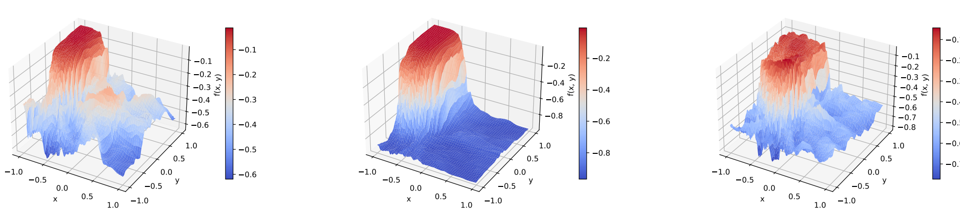

Figure 1: Examples of three objective functions f_1, f_2, f_3 corresponding to the cross-validation error of a SVM as a function of the hyper-parameters on three different datasets where the x-axis denotes the regularization parameter and y-axis denotes the bandwidth parameter in log-scale.

Algorithm 1 MELBA (MEta bLack Box optimizAtion)

Require: Class function distribution $\mathcal{F}$; BBO budget N; meta iterations M

1: Initialize belief function B_{θ_T} and policy π_{θ_P}
2: **for** $i = 1$ to M **do** ▷ Outer meta-loop: learning the solver
3: $\quad f_i \sim \mathcal{F}$, $o_0 = \{\}$, $f_{\mathtt{max}} = -\infty$, $\tau = \{\}$
4: $\quad$ **for** $n = 0$ to $N - 1$ **do** ▷ Inner meta-loop: applying the solver
5: $\quad\quad z_n^b = B_{\theta_T}(o_n)$ ▷ Compute latent belief state
6: $\quad\quad x_n \sim \pi_{\theta_P}(\cdot|z_n^b)$ ▷ Sample a candidate solution
7: $\quad\quad f_{\mathtt{max}} = \mathtt{max}(f_{\mathtt{max}}, f_i(x_n))$ ▷ Evaluate and update max observed so far
8: $\quad\quad c_n = f^* - f_{\mathtt{max}}$ ▷ Compute reward
9: $\quad\quad o_{n+1} = o_n \cup \{(x_n, f_i(x_n)\}$ ▷ Update information gathered so far
10: $\quad\quad \tau = \tau \cup \{(o_n, x_n, r_n, o_{n+1})\}$ ▷ Update RL trajectory
11: $\quad$ **end for**
12: $\quad$ Compute PPO objective $\mathcal{L}_{\text{RL}}(\tau)$
13: $\quad$ Update π_{θ_P}, B_{θ_T} according to $\mathcal{L}_{\text{RL}}(\tau)$ ▷ Gradient descent steps
14: **end for**

3 Meta-learn problem-specific black-box solvers through deep reinforcement learning

Using Transformers to define a class of learnable algorithms. First, in order to solve (3), we define a large class of algorithms Alg through the use of Transformers. More precisely, at any time $n \leq N$, based on the set of previously evaluated points $o_n = (x_1, f(x_1), \ldots, (x_n, f(x_n)))$, an algorithm A selects the next evaluation points x_{n+1} based on the history o_n. One general way to formalize this process is to consider algorithms (without loss of generality) that sample the next points according to a Gaussian distribution with learnable parameters:

$$x_{n+1} \sim \pi_{\theta_P}(\cdot|z_n^b) = \mathcal{N}(\mu_{\theta_P}(z_n^b), \Sigma_{\theta_P}) \,, \tag{4}$$

where z_n^b denotes a latent representation of the current observations o_n, and $\mu_{\theta_P}, \Sigma_{\theta_P}$ are two learnable functions (e.g. feed-forward networks) that represent the mean and covariance matrix of the gaussian. Inspired by (Xiang and Foo, 2021), the idea here is to learn a latent representation z_n^b of the current information o_n informative enough such that the more evaluated points, the less uncertain the belief. To take the sequential information o_n into account, a natural approach is to consider a state-of-the-art sequential model such as the Transformer architecture (Vaswani et al., 2017) that allows us to compute pairwise interactions between the collected evaluation points. Formally, the parametrized belief function B_{θ_T} embeds the set of observations $o_n = \{(\mathbf{x}_i, f(\mathbf{x_i}))\}_{i\in\{1,\ldots n\}}$ in a latent space through several layers of Transformer encoders. Finally, the final representations are averaged to provide the belief latent representation $z_n^b = B_{\theta_T}(o_n)$ which is used as the input of the algorithm π_{θ_P} through μ_{θ_P} instead of the raw observation o_n.

Minimizing the meta-loss using reinforcement learning. Second, we focus on minimizing the loss (3). The first potential idea is to directly differentiate the loss (3) that assesses the performance of the learned algorithm, which have been explored in previous works (TV et al., 2019; Chen et al., 2016). However, to differentiate (3), it is necessary for the functions $f_1, \ldots, f_M$ in $\mathcal{F}$ to be differentiable, which greatly restricts the range of applications of the meta-approach to black-box functions. With these considerations in mind, RL appears as a natural way to optimize this non-differentiable loss. Indeed, similarly to RL problems, our framework relies on optimizing a parametrized algorithm (here the policy π_{θ_P}) to minimize the observed error (here the reward (1)) that is not necessarily differentiable. Our algorithm is thus optimized in a RL fashion by computing the standard surrogate loss $\mathcal{L}_{\text{RL}}$ of the Proximal Policy Optimization Schulman et al. (2017) (PPO) and performing gradient descent, learning together the Transformer parameters θ_T and the policy parameters θ_P. As classical meta-learning approaches, our algorithm called MELBA (Algorithm 1) consists in two optimization loops: an outer meta-loop optimizing the parameters of the learned solver by gradient descent (line 2); an inner meta-loop optimizing the black-box function with the current parametrized solver (line 4). More precisely, the outer loop contains the optimization of a meta-loss that is designed to learn the optimization algorithm over $\mathcal{F}$.

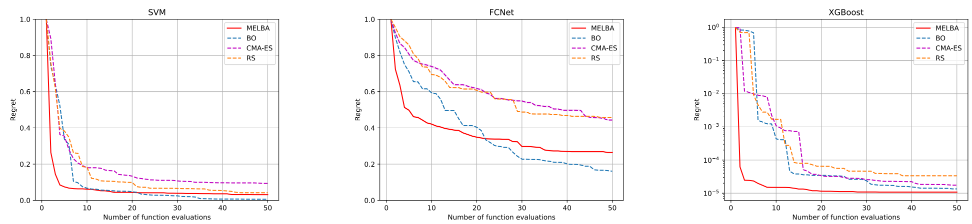

Figure 2: Performance of the algorithms on the hyper-parameter optimization problems.

4 Experimental evaluation

We considered three baselines. **Random Search (RS)** is the standard random method which consists in evaluating the objective function on a series of points randomly sampled over the input space. While simple, RS has been shown to be difficult to outperform on some black-box problems. **CMA-ES** is a state-of-the-art evolutionary algorithm that samples the next evaluation points according to a multivariate normal distribution. **Bayesian Optimization (BO)** builds a surrogate model of the black-box function and chooses points according to an acquisition function that trades off uncertainty and best known regions. It is one of the most successful methods for hyper-parameter tuning. For **MELBA**, the algorithm presented in this paper, the policy is parametrized by a 512-dimension latent space, a four-layer Transformer encoder with four heads followed by two fully connected layers. Details regarding the experiments are provided in the Appendix.

Test problems. The goal of MELBA is to learn specific hyper-parameter optimization algorithms from a training set of trained Machine Learning models on different datasets to generalize to unseen datasets. We compared MELBA to the baselines on three test problems. **Support Vector Machines (SVM).** This two-dimensional problem consists of learning an algorithm that is specifically tailored to optimize the hyper-parameters of the SVM. More precisely, the parameters to optimize are the regularization parameter $10\log(C)$ and the bandwidth parameter $10\log(\gamma)$ of the Radial Basis Function (RBF) kernel for any dataset. The parameters were rescaled to be in $\mathcal{X} = [-1, 1]^2$. **Fully Connected Networks (FCNet).** This six-dimensional problem consists of finding the parameters that define the architecture of dense neural network to perform classification. The different parameters are the learning rate, the batch size, the width of the first layer, the width of the second layer, the dropout rates for the first and second layer. The parameters were rescaled to be in $\mathcal{X} = [-1, 1]^6$. **XGBoost.** This eight-dimension problem consists of finding the hyper-paramaters of an XGBoost (Chen and Guestrin, 2016) regardless of the dataset. The hyper-parameters are: the learning rate, gamma, res_alpha, reg_lambda, n_estimators, sub_sample, max_depth, min_child_weight. Again, all the parameters were rescaled to be in $\mathcal{X} = [-1, 1]^8$. For all these problems, we used the benchmarking suite for hyper-parameter optimization called PROFET (Klein et al., 2019). PROFET uses generative models to create realistic HPO tasks. These generative models allow to query the value at some point in the space of the black-box functions $f_1, \dots, f_M$ (that represents the metric of the trained model as a function of its hyper-parameters) corresponding to the above problems. Each instance of MELBA has been trained on $M = 970$ test problems for each task and we considered a budget of $N = 50$ evaluations.

Performance metrics and discussion.

To measure the performance of the algorithm on a given class of functions $\mathcal{F}$, we computed the empirical regret $\text{Regret}(A, \mathcal{F}, n) = \frac{1}{M}\sum_{j=1}^{M} r(A, f_j, n)$ for each iteration $n = 1 \dots N$ where $f_1, \dots, f_M$ are randomly sampled from the family of functions $\mathcal{F}$ with $M = 30$ and unseen during the training phase. Results are displayed in Figure 2. MELBA shows high-performance compared to the baselines on all tasks. It outperforms CMA-ES on all tasks. It outperforms RS on all tasks but the SVM one where it has comparable performance. MELBA outperforms BO on the XGBoost task while having comparable performances on the SVMs and FCNet ones. We observe that MELBA-learned solvers find a significantly better solution than the competitors within only 10 function evaluations. Then, MELBA-solvers are slightly overtaken by BO on the SVMs and FCNet benchmarks. In the case of XGBoost functions, the MELBA-solver is very efficient and reaches a better value in 20 steps than the results reached by competitors in 50 steps. These promising results outline the relevance of learning task-specific solvers with Meta-learning for hyper-parameter optimization.

References

Alarie, S., Audet, C., Gheribi, A. E., Kokkolaras, M., and Le Digabel, S. (2021). Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization*, 9:100011.

Blank, J. and Deb, K. (2020). pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509.

Chen, T. and Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA. ACM.

Chen, Y., Hoffman, M. W., Colmenarejo, S. G., Denil, M., Lillicrap, T. P., Botvinick, M., and de Freitas, N. (2016). Learning to learn without gradient descent by gradient descent.

Head, T., MechCoder, Louppe, G., Shcherbatyi, I., fcharras, Vinícius, Z., cmmalone, Schröder, C., nel215, Campos, N., Young, T., Cereda, S., Fan, T., rene rex, Shi, K. K., Schwabedal, J., carlosdanielcsantos, Hvass-Labs, Pak, M., SoManyUsernamesTaken, Callaway, F., Estève, L., Besson, L., Cherti, M., Pfannschmidt, K., Linzberger, F., Cauet, C., Gut, A., Mueller, A., and Fabisch, A. (2018). scikit-optimize/scikit-optimize: v0.5.2.

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

Klein, A., Dai, Z., Hutter, F., Lawrence, N., and Gonzalez, J. (2019). Meta-surrogate benchmarking for hyperparameter optimization.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. (2017). Evolution strategies as a scalable alternative to reinforcement learning.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

TV, V., Malhotra, P., Narwariya, J., Vig, L., and Shroff, G. (2019). Meta-learning for black-box optimization.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need.

Waring, J., Lindvall, C., and Umeton, R. (2020). Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial Intelligence in Medicine*, 104:101822.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Xiang, X. and Foo, S. (2021). Recent advances in deep reinforcement learning applications for solving partially observable markov decision processes (pomdp) problems: Part 1fundamentals and applications in games, robotics and natural language processing. *Machine Learning and Knowledge Extraction*, 3(3):554–581.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R., and Smola, A. (2017). Deep sets.

A Time Complexity of MELBA vs BO

The time complexity of one iteration of MELBA in comparison with Bayesian Optimization is presented in table 1. We show that MELBA's time complexity is quasi-constant with the dimension and that it is from 70 times faster in dimension 2 to more than 150 times faster in dimension 8. This is due to the inner maximization of the acquisition function in BO. Furthermore, time complexity is almost constant with the dimension in our approach while getting higher and higher in BO. This is a significant advantage of our approach.

Benchmark	Dimension	MELBA	BO
SVM	2	**0.08**	5.66
FCNet	6	**0.08**	10.68
XGBoost	8	**0.08**	12.51

Table 1: Average time of an iteration (in seconds) of MELBA instances vs Bayesian Optimization as a function of the dimension

B Details of the experimental section

The PROFET HPO benchmark (Klein et al., 2019) has been introduced to allow more reproducible research in AutoML. It avoids heavy and time-consuming training computations when testing HPO methods. Each HPO task is created thanks to some generative model fitted on some results of true training datasets. The generative model, then, allows to infer the value of the targeted metric (MSE or classification error) on unseen and continuous values of hyperparameters.

C Hyperparameter validation

In this section, we provide the technical details of our implementation and the hyperparameters we used for our experiments.

C.1 Proximal Policy Optimization algorithm and Transformer encoder

We validated the hyperparameters of our framework (both PPO and the Transformer encoders) by random search on each benchmark over a discrete grid of predefined values. Grids are presented in tables 2 and 3 for PPO and the attention network architecture respectively. We train all the models on 1000000 time-steps.

Name	Values
Learning rate	$3 \times 10^{-3}, 3 \times 10^{-4}, 1 \times 10^{-4}, 7.5 \times 10^{-5}, 3 \times 10^{-5}$
N steps	$32, 64, 128, 256, 512, 1024, 2048, 4092$
Batch size	$4, 8, 16, 32, 64, 128, 256$
Epochs	$10, 20, 30$
Gamma	$0.8, 0.9, 0.99$
Clip range	$0.1, 0.2, 0.3$
Entropy coefficient	$0.01, 0.001, 0$
Value function coefficient	$0.5, 0.7, 1$

Table 2: Grid over PPO hyperparameters.

Tables 4 and 5 provide the hyperparameters selected for each benchmark for the PPO algorithm and the network architecture respectively:

C.2 Baselines

To run CMA-ES, we used the same values as in Klein et al. (2019), *i.e.*, $\sigma = 0.6$ as the initial standard deviation and 10 for the population size. We used the pymoo library (Blank and Deb, 2020)

Name	Values
Encoder layers	$1, 2, 4, 8$
Fully connected layers	$1, 2, 4, 6$
Hidden heads	$1, 2, 4, 8$
Feedforward dimension	$32, 64, 128, 256, 512, 1024$
Latent representation dimension	$32, 64, 128, 256, 512, 1024$
Dropout	$0, 0.1, 0.2, 0.4, 0.8$

Table 3: Grid over the Transformer encoders hyperparameters.

Benchmark	Learning rate	N steps	Batch size	Epochs	Gamma	Clip range	Entropy Coefficient	Value function coefficient
SVM	0.0001	512	32	10	0.99	0.1	0	0.7
‘ FCNet	0.0003	512	32	10	0.99	0.1	0	0.7
XGBoost	0.000075	512	32	10	0.99	0.1	0	0.7

Table 4: PPO hyperparameters for each benchmark

Benchmark	Encoder layers	Fully connected layers	Hidden heads	Feedforward dimension	Latent representation dimension	Dropout
SVM	4	2	4	256	256	0.1
FCNet	4	2	4	256	256	0.1
XGBoost	4	2	4	512	512	0.1

Table 5: Transformer encoders hyperparameters for each benchmark

implementation. For the Bayesian Optimization baseline, we used the Expected Improvement as the acquisition function and we used the implementation of Scikit-Optimize (Head et al., 2018).

D Computational details

All the experiments, both training and inference, were done on a Tesla P100 GPU. During training, we used multiprocessing with 8 CPUs to run 8 parallel environments.

E Ablation studies

In this section, we present some ablation studies on the impact of the Transformer architecture. For this purpose, we compare the results obtained on the different benchmarks with our architecture against Deep Set (Zaheer et al., 2017) which consists of shared fully connected layers with every element in the set of evaluated points. The experimental protocol is the same as in the main results. The Deep Set model consists of 4 fully connected layers with 512 neurons. The results are shown in figures 3 for the HPO benchmarks. These demonstrate the importance of the attention mechanism since the Transformer encoders outperform Deep Set consistently.

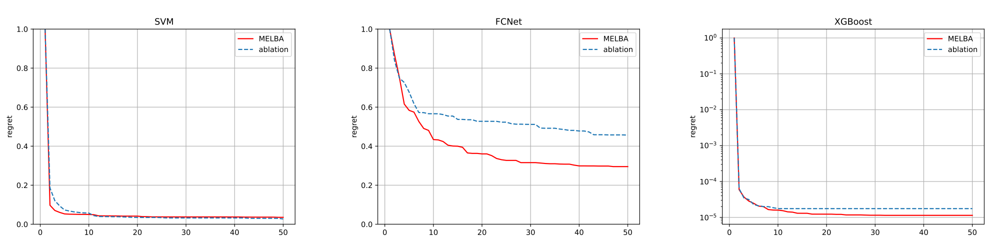

Figure 3: Ablation study on the PROFET HPO benchmarks.