
POLICY-BASED OPTIMIZATION: SINGLE-STEP POLICY GRADIENT METHOD SEEN AS AN EVOLUTION STRATEGY

A PREPRINT

J. Viquerat*

MINES Paristech, CEMEF

PSL - Research University

jonathan.viquerat@mines-paristech.fr

R. Duvigneau

INRIA Sophia Antipolis Méditerranée

ACUMES project-team

P. Meliga

MINES Paristech, CEMEF

PSL - Research University

A. Kuhnle

University of Cambridge

E. Hachem

MINES Paristech, CEMEF

PSL - Research University

September 29, 2021

Abstract

1 This research reports on the recent development of black-box optimization methods
2 based on single-step deep reinforcement learning (DRL) and their conceptual similar-
3 ity to evolution strategy (ES) techniques. It formally introduces policy-based opti-
4 mization (PBO), a policy-gradient method that relies on a policy network to describe
5 the density function of its forthcoming evaluations, and uses covariance estimation to
6 steer the policy improvement process in the right direction. The specifics of the PBO
7 algorithm are detailed, and the connection to evolutionary strategies (especially co-
8 variance matrix adaptation evolutionary strategy) is discussed. Relevance is assessed
9 by benchmarking PBO against classical ES techniques on analytic functions mini-
10 mization problems, and by optimizing various parametric control laws intended for
11 the Lorenz attractor. Given the scarce existing literature on the topic, this contribu-
12 tion definitely establishes PBO as a valid, versatile black-box optimization technique,
13 and opens the way to multiple future improvements building on the inherent flexibility
14 of the neural networks approach.

15 **Keywords** Deep reinforcement learning; Artificial neural networks; Evolution strategies; CMA-ES;
16 Black-box optimization; Lorenz attractor; Parametric control law

17 1 Introduction

18 During the past decade, machine learning methods, and more specifically deep neural network (DNN),
19 have achieved great success in a wide variety of domains. State-of-the-art neural network architectures
20 have reached astonishing performance levels in a variety of tasks, *i.e.*, image classification [1, 2], speech
21 recognition [3] or generative tasks [4], to name a few. With generalized access to GPU computational
22 resources through cheaper hardware or cloud computing, such advances have opened the path to a
23 revolution of the reference methods in these domains, at both academic and industrial levels.

24 With neural networks quickly becoming pervasive in a broad range of domains, significant progress has
25 been made in solving challenging decision-making problems by deep reinforcement learning (DRL), an
26 advanced branch of machine learning that couples DNNs and reinforcement learning (RL) algorithms.

*Corresponding author

27 The ability to use high-dimensional state spaces and to exploit the feature extraction capabilities of
28 DNNs has proven decisive to lift the obstacles that had long hindered classical RL methods. In return,
29 this yielded unprecedented efficiency in games [5, 6, 7] and in several scientific disciplines such as
30 robotics [8], language processing [9], although a tremendous potential also exists for applying DRL to
31 real-life applications, including autonomous cars [10, 11] or data center cooling [12].

32 Although neural networks are regularly used in optimization problems, they are most often exploited
33 as trained surrogates for the actual objective function [13, 14], and have long been mostly left out of
34 the central optimization process, with the exception of a handful of studies [15, 16, 17]. This trend has
35 been changing lately, as tweaked versions of classical DRL policy gradient algorithms have begun to
36 be used as black-box optimizers [18], the underlying idea being that a DNN can learn to map the same
37 initial state to an optimal set by interacting only once per episode with its environment, if the policy
38 to be learnt is independent of state (hence, single-step episodes, and by extension, single-step DRL).
39 Such an approach has been speculated to hold a high potential for reliable optimization of complex
40 systems. Nonetheless, it remains to be analyzed in full depth, as feasibility has just been assessed in a
41 computational fluid mechanics (CFD) context, including shape optimization [18], drag reduction [19]
42 and conjugate heat transfer control [20] (a similar concept of "stateless DRL" has been early sketched
43 in [21] for validation purpose, but was not pursued).

44 This research formally introduces policy-based optimization (PBO), a novel single-step DRL algorithm
45 that shares strong similarities with evolution strategies (ES). The objective is twofold: first, to deliver
46 several major improvements over our previous PPO-1 algorithm, by adopting key heuristics from the
47 covariance matrix adaptation evolution strategy (CMA-ES); second, to shape the capabilities of the
48 method and to provide performance comparison against canonical ES algorithms on textbook and
49 applied cases. One of the key novelty lies in the use of three separate neural networks to learn the
50 mean, variance and correlation parameters of a multivariate normal search distribution, yielding a
51 powerful, flexible optimization method (without anticipating the results, PBO compares very well
52 with CMA-ES, which is all the more promising since new algorithms cannot be expected to reach right
53 away the level of performance of their more established counterparts). In comparison, PPO-1 updates
54 the mean and variance (the same for all variables) from a single neural network, which can prematurely
55 shrink the exploration variance. While there have been previous attempts to similarly improve the
56 convergence properties of classical DRL algorithm by drawing inspiration from ES [21], our literature
57 review did not reveal any other study considering the generation of valid full covariance matrices from
58 neural network outputs.

59 The organization of the remaining of the paper is as follows: section 2 provides the needed background
60 on policy gradient reinforcement learning and evolutionary strategies. The policy-based optimization
61 (PBO) algorithm is introduced in section 3, where we thoroughly examine how to generate valid full
62 covariance matrices from neural network outputs, and point out key similarities and differences with
63 respect to CMA-ES.² In section 4, PBO is benchmarked against standard ES algorithms on textbook
64 optimization problems of analytic functions minimization. Finally, section 5 uses PBO to optimize
65 a parametric control law for the Lorenz attractor, a well-known reduced version of Rayleigh–Bénard
66 convection.

67 2 Preliminaries

68 2.1 Neural networks

69 A neural network (NN) is a collection of artificial neurons, *i.e.*, connected computational units that
70 can be trained to approximate arbitrarily well the mapping function between input and output spaces
71 [22]. Each connection provides the output of a neuron as an input to another neuron. Each neuron
72 performs a weighted sum of its inputs, to assign significance to the inputs with regard to the task the
73 algorithm is trying to learn. It then adds a bias to better represent the part of the output that is
74 actually independent of the input. Finally, it feeds an activation function that determines whether and
75 to what extent the computed value should affect the ultimate outcome. A fully connected network is
76 generally organized into layers, with the neurons of one layer being connected solely to those of the
77 immediately preceding and following layers. The layer that receives the external data is the input

²The base code used to produce all results documented in this paper is available via a dedicated github repository [42].

78 layer, the layer that produces the outcome is the output layer, and in between them are zero or more
79 hidden layers.

80 The design of an efficient neural network requires well-chosen nonlinear activation functions, together
81 with a proper optimization of the weights and biases, to minimize the value of a loss function suitably
82 representing the quality of the network prediction. The network architecture (e.g., type of network,
83 depth, width of each layer), the meta-parameters (*i.e.*, parameters whose value cannot be estimated
84 from data, e.g., optimizer, learning rate, batch size) and the quality/size of the dataset are other key
85 ingredients to an efficient learning, that must be carefully crafted to the intended purpose. For the
86 sake of brevity, the reader is referred to [23] for an extended presentation of this topic.

87 2.2 Reinforcement learning

88 Reinforcement learning (RL) is a subset of machine learning in which an agent learns to solve decision-
89 making problems by earning rewards through trial-and-error interaction with its environment. It
90 is mathematically formulated as a Markov decision process in which the agent observes the current
91 environment state s_t , takes an action a_t that prompts the reward received r_t and the transition to the
92 next state s_{t+1} , and repeats until the agent is unable to increase some form of cumulative reward. In
93 practice, this is framed as an optimization problem of maximizing the discounted reward cumulated
94 over a horizon T , defined as:

$$R(\tau) = \sum_{t=0}^T \gamma^t r_t, \quad (1)$$

95 where $\tau = (s_0, a_0, s_1, a_1, \dots, s_T, a_T)$ is a trajectory of states and actions, and $\gamma \in [0, 1]$ is a discount
96 factor that weights the relative importance of present and future rewards.

97 2.3 Policy gradient RL methods

98 In policy gradient methods, the agent behavior is modeled after a stochastic policy $\pi_\theta(s, a)$, *i.e.* a
99 probability distribution over actions given states. The expected discounted cumulative reward $J(\theta)$ is
100 maximized by gradient ascent on the policy parameters θ , updating at each iteration by a fixed-size
101 step proportional to the policy gradient, whose expression derived in [24] in the context of "vanilla"
102 policy gradient reads:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \nabla_\theta \log \pi_\theta(s_t, a_t) R(\tau) \right]. \quad (2)$$

103 In a deep reinforcement learning context (deep RL or DRL), the policy is represented by a deep neural
104 network whose weights and biases serve as free parameters to be optimized. To this end, a stochastic
105 gradient algorithm is used to perform network updates from the policy loss:

$$L(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \log \pi_\theta(s_t, a_t) R(\tau) \right], \quad (3)$$

106 whose gradient is equal to $\nabla_\theta J(\theta)$ (since the gradient operator in (2) acts only on the log-policy term),
107 and is computed with the back-propagation algorithm [25] with respect to each weight and bias by the
108 chain rule, one layer at the time from the output to the input layer.

109 Multiple refinements varying in cost, complexity and purpose have been proposed to steer the policy
110 improvement process in the right direction, whether it be by balancing the trade-off between bias and
111 variance (actor-critic [26], generalized advantage estimate [27]) or by preventing the destructively large
112 policy updates that can cause the agent to fall off the cliff and to restart from a poorly performing
113 state with a locally bad policy (trust-region policy optimization, proximal policy optimization [28]).
114 We shall not elaborate further on this matter, as the present implementation uses a variant of the
115 vanilla policy gradient update. Hence, the interested reader is instead referred to [29] and references
116 therein.

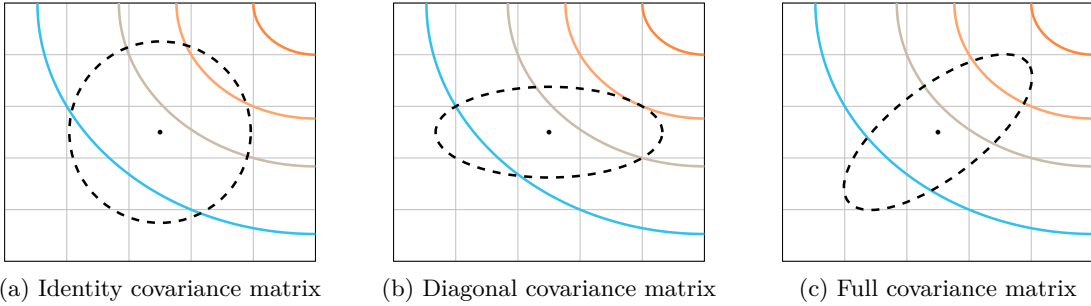


Figure 1: Iso-density lines for multivariate normal laws with identity, diagonal and full covariance matrices.

117 2.4 Evolution strategies

118 Evolution strategies (ES) are another family of stochastic search algorithm that can learn an optimal
 119 parametrization by emulating organic evolution principles, without knowledge of the performance gra-
 120 dient. At each iteration g (called generation), the algorithm samples λ candidate solutions (x_1, \dots, x_λ)
 121 from a multivariate normal distribution $\mathcal{N}(\mathbf{m}^g, \mathbf{C}^g)$ with mean \mathbf{m}^g and covariance matrix \mathbf{C}^g , evalu-
 122 ates the cost function at the candidate solutions, and uses a weighted recombination of the μ best
 123 individuals to update the search distribution for the next generation. Simply put, the mean is pulled
 124 into the direction of the best performing candidates, while the covariance update aims to align the den-
 125 sity contour of the sampling distribution with the contour lines of the objective function and thereby
 126 the direction of steepest descent. The range of possible models corresponds to various degrees of sophis-
 127 tication. For instance, (μ, λ) -ES is a rudimentary algorithm relying on identity covariance matrices,
 128 *i.e.*, it assumes all variables to have the same variance and to be uncorrelated, which in turn defines an
 129 isotropic region of sampling for the next generation (see figure 1a). Conversely, the covariance matrix
 130 adaptation evolutionary strategy (CMA-ES, considered state-of-the-art in evolutionary computations)
 131 uses a full covariance matrix to accelerate convergence toward the optimum by exploiting anisotropy in
 132 the steepest descent direction (see figure 1c). Another key aspect lies in the structure of the CMA-ES
 133 covariance matrix update:

$$\mathbf{C}^{g+1} \leftarrow (1 - c_1 - c_\mu)\mathbf{C}^g + c_\mu\mathbf{C}_\mu + c_1\mathbf{C}_1 \quad (4)$$

134 where the first term represents a soft update from the current covariance matrix, and c_1 and c_μ are
 135 learning rates set by well-established heuristics and associated to two types of updates termed *rank-*
 136 *1* and *rank- μ* . The rank- μ update includes information about the best individuals of the current
 137 generation, while the rank-1 update adds correlation information across consecutive generations via a
 138 so-called evolution path storing the average update direction (in a way such that correlated updates sum
 139 up but decorrelated updates cancel each other out). These three contributions combined ultimately
 140 allow CMA-ES to fast search from limited populations of individuals at each generation, without
 141 compromising the evaluation of the next covariance matrix, as thoroughly described in [30].

142 3 Policy-based optimization (PBO)

143 We review below the main features of our proposed policy-based optimization (PBO) algorithm, and
 144 point out the key conceptual similarities and differences with respect to the methods introduced in
 145 section 2. In order to provide common ground between all approaches, we refer from now on to each
 146 new set of evaluation as a *generation* g , and to each evaluation within a generation as an *individual*.
 147 Also, we denote by n_i the number of individuals evaluated at each generation (*i.e.* the number of
 148 parallel environments used to collect rewards before performing a network update) and by d the search
 149 space dimension (*i.e.* the dimension of the action required by the environment).

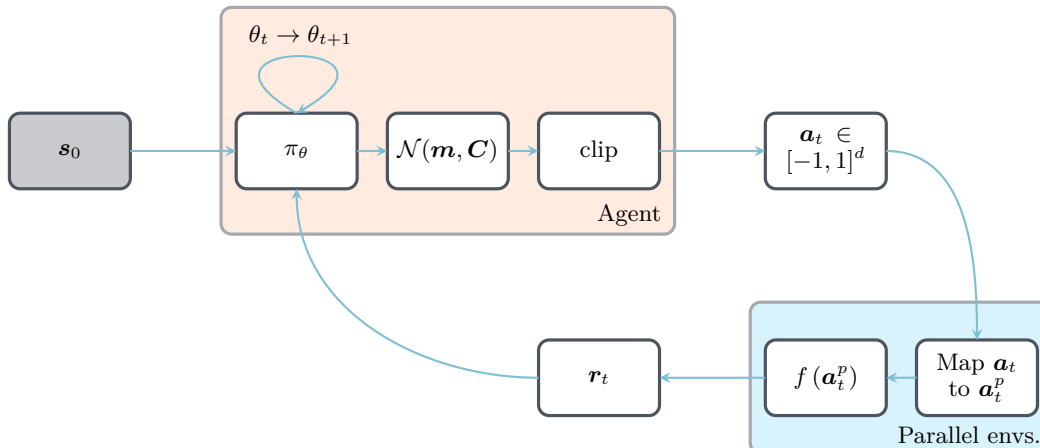


Figure 2: **Action loop for the PBO method.** At each generation, the same input state \mathbf{s}_0 is provided to the agent, that draws a set of actions $\mathbf{a} \in [-1, 1]^d$ from the current probability distribution function, with d the problem dimensionality. The actions are distributed to several parallel environments, and mapped to physical ranges \mathbf{a}^P . The parallel environments then evaluate the cost function f at the physical actions, and returns a set of rewards \mathbf{r} measuring the quality of the actions taken. Once a sufficient amount of state-action-reward triplets has been collected, the network parameters are updated from the policy loss (5). The process is repeated until convergence.

150 3.1 Single-step deep reinforcement learning

151 Policy-based optimization (PBO) is a single-step policy gradient RL algorithm whose premise is that
 152 it is enough that the agent interacts with the environment only once per episode (defined as one
 153 instance of the scenario in which the agent takes actions, hence single-step) if the policy to be learnt
 154 is independent of state, *i.e.* $\pi_\theta(s, a) \equiv \pi_\theta(a)$. This is notably the case in optimization and open-loop
 155 control problems (the policy in closed-loop control problems conversely depends on states thus requires
 156 multiple interactions per episode).

157 The line of thought is as follows: where a standard policy gradient algorithm seeks the optimal θ^*
 158 such that following π_{θ^*} maximizes the discounted cumulated reward over an episode, PBO seeks the
 159 optimal θ^* such that $\mathbf{a}^* = \pi_{\theta^*}(\mathbf{s}_0)$ maximizes the instantaneous reward, with \mathbf{s}_0 being some input state
 160 (usually a constant vector) consistently fed to the agent for the optimal policy to eventually embody
 161 the optimal transformation from \mathbf{s}_0 to \mathbf{a}^* . The agent initially implements a random policy determined
 162 by its initial set of parameters θ_0 , after what it gets only one attempt per episode at finding the optimal.
 163 This is illustrated in figure 2, showing the agent draw a population of actions from the current policy,
 164 and being incentivized to update the policy parameters for the next population of actions to yield
 165 larger rewards. A direct consequence is that PBO uses smaller policy networks (compared to usual
 166 agent networks found in other DRL contributions), because the agent is not required to learn a complex
 167 state-action relation, but only a transformation from a *constant* input state to a given action.

168 3.2 Gradient ascent update rule

169 In practice, PBO draws actions from a probability density function. Here, we use a d -dimensional
 170 multivariate normal distribution $\mathcal{N}(\mathbf{m}, \mathbf{C})$ with mean \mathbf{m} and full covariance matrix \mathbf{C} . As shown in
 171 figure 3, three independent neural networks are used to output the necessary mean, standard deviation,
 172 and correlation information, using hyperbolic tangent and sigmoid activation functions on the output
 173 layers to constrain all values in their respective adequate ranges (see section 3.3 for more details).
 174 Actions are then drawn in $[-1, 1]^d$ by clipping (a series of numerical experiments indicates that soft-
 175 limiting transfer functions such as hyperbolic tangent or soft clipping are generally not beneficial
 176 and yield, in certain cases, slow convergence and numerical instabilities), before being mapped to
 177 their relevant physical ranges \mathbf{a}^P (a step deferred to the environment as being problem-specific), as
 178 illustrated in figure 2. Finally, the Adam algorithm [31] runs stochastic gradient ascent on the policy
 179 parameters using the modified loss:

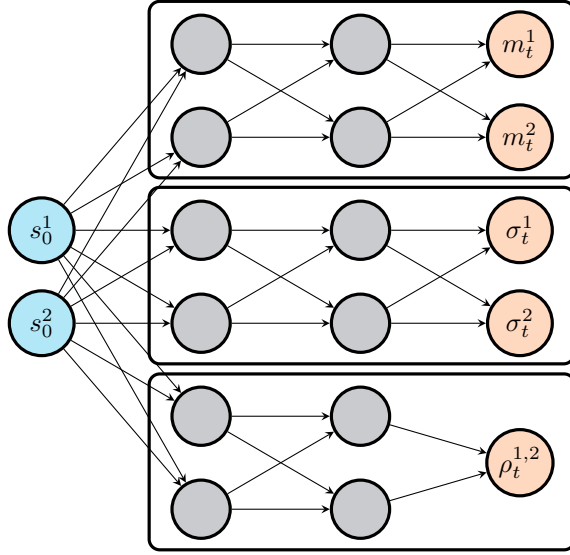


Figure 3: **Policy networks used in PBO to map states to policy.** Three separate networks are used for the prediction of mean, standard deviation, and correlation parameters. All activation functions are hyperbolic tangents, except for the output layers of the σ and ρ networks, which uses sigmoid (please refer to section 3.3 for additional details). Orthogonal weights initialization is used throughout the networks, with a unit gain for all layers except the output layers, for which the gain is set to 1×10^{-2} . In practice, all three networks are trained separately.

$$L(\theta) = \mathbb{E}_{a \sim \pi_\theta} \left[\log \pi_\theta(a|s_0) \max \left(\frac{r - \bar{r}}{\hat{r}}, 0 \right) \right], \quad (5)$$

180 where \bar{r} (resp. \hat{r}) is the reward average (resp. standard deviation) over the current generation. and
 181 the PBO loss is thus formally identical to (3), except for the clipped generation-wise whitened reward
 182 substituted for the discounted cumulative reward. The rationale for this choice is as follows: as is
 183 customary in DRL, the discounted cumulative reward is approximated by the advantage function, that
 184 measures the improvement (if positive, otherwise the lack thereof) associated with taking action a in
 185 state s compared to taking the average over all possible actions. Because a PBO trajectory consists of
 186 a single state-action pair (hence (5) drops the sum over t), the discount factor can be set to $\gamma = 1$, in
 187 which case the advantage reduces to the reward, as further explained in [19]. The present normalization
 188 to zero mean and unit standard deviation introduces bias but reduces variance, and thus the number of
 189 actions needed to estimate the expected value. Finally, the max allows discarding negative-advantage
 190 actions, that may destabilize learning when performing multiple mini-batch gradient steps using the
 191 same data (as each step drives the policy further away from the sampled actions).

192 3.3 Generating valid covariance matrices from neural network outputs

193 Matrices representing correlations between variables must satisfy four basic properties to bear physical
 194 significance: (i) all entries must be in $[-1, 1]$ (nothing goes beyond perfect correlation or perfect
 195 anticorrelation), (ii) all diagonal entries must be equal to 1 (a variable is always perfectly correlated
 196 with itself), (iii) the matrix must be symmetric (correlation between variables i and j is equal to
 197 correlation between j and i), and (iv) the matrix must be positive semidefinite (PSD, the variance of
 198 a weighted sum of the random variables must be positive). It follows that the above naive approach
 199 consisting in having a neural network directly output a set of correlation parameters in adequate
 200 range is vowed to fail, as there is no guarantee whatsoever that the so-obtained matrix will be PSD.
 201 In addition, while it is possible on paper to have the neural network repeatedly output correlation
 202 coefficients until a PSD matrix is obtained (which amounts to implementing the classical rejection
 203 sampling method), this quickly becomes inefficient as the chances of finding a valid matrix are very
 204 low for $d > 3$.

205 PBO overcomes this issue using hypersphere decomposition, a method rooted in risk management
 206 theory, that generates valid correlation matrices from a set of angular coordinates on a hypersphere of
 207 unit radius [32, 33]. The reader interested in a detailed and comprehensive presentation of the method
 208 is referred to [34]. We shall just mention here that the method parameterizes a lower triangular
 209 elementary matrix \mathbf{B}_d with entry:

$$b_{ij} = \begin{cases} 1 & \text{for } i = j = 1 \\ \cos \varphi_{ij} & \text{for } i > 1, j = 1 \\ \cos \varphi_{ij} \prod_{k=1}^{j-1} \sin \varphi_{ik} & \text{for } i > 1, j < i \\ \prod_{k=1}^{j-1} \sin \varphi_{ik} & \text{for } i > 1, j = i \\ 0 & \text{for } j > i \end{cases} \quad (6)$$

210 from a set of so-called correlative angles $\varphi \in [0, \pi]^D$, with $D = \frac{d(d-1)}{2}$ (hence in same number as the
 211 correlation parameters). For instance the matrix for $d = 4$ reads:

$$\mathbf{B}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \cos \varphi_{2,1} & \sin \varphi_{2,1} & 0 & 0 \\ \cos \varphi_{3,1} & \cos \varphi_{3,2} \sin \varphi_{3,1} & \sin \varphi_{3,2} \sin \varphi_{3,1} & 0 \\ \cos \varphi_{4,1} & \cos \varphi_{4,2} \sin \varphi_{4,1} & \cos \varphi_{4,3} \sin \varphi_{4,2} \sin \varphi_{4,1} & \sin \varphi_{4,3} \sin \varphi_{4,2} \sin \varphi_{4,1} \end{bmatrix} \quad (7)$$

212 The product of this matrix with its transpose is then guaranteed to be a valid correlation matrix, as it
 213 is symmetric and PSD by construction, with all entries in $[-1, 1]$ (since all B_{ij} are products of cosine
 214 and sine functions) and unit diagonal [35].

215 The retained procedure to efficiently doctor neural network outputs into valid parameterization of a
 216 multivariate normal distribution is thus as follows: the first network outputs the mean \mathbf{m} in $[-1, 1]^d$
 217 using a hyperbolic tangent activation function on the output layer. The second network outputs the
 218 standard deviations $\boldsymbol{\sigma}$ in $[0, 1]^d$ using a sigmoid activation function on the output layer. Finally, the
 219 third network outputs a set of coefficients $\boldsymbol{\rho}$ in $[0, 1]^D$, also using a sigmoid activation function on
 220 the output layer. Those are mapped into correlative angles $\varphi = \pi \boldsymbol{\rho}$ and assembled into the above
 221 elementary matrix \mathbf{B}_d , after which the covariance matrix is constructed as:

$$\mathbf{C} = \mathbf{S} (\mathbf{B}\mathbf{B}^t) \mathbf{S}, \quad (8)$$

222 with $\mathbf{S} = \text{diag}(\boldsymbol{\sigma})$.

223 3.4 Off-policy updates

224 Accurately computing the expected value in the policy loss (5) requires sampling a large number of
 225 state-action-reward triplets before the algorithm can proceed to update the agent parameters. At each
 226 generation, a set of actions drawn from the current policy π_θ is thus distributed to n_i environments
 227 running in parallel, each of which computes a reward associated to its input action, and provides it back
 228 to the agent. This can repeat until the agent has collected a sufficient number of state-action-reward
 229 triplets. Still, in many cases, it is not tractable to use a large value of n_i because computing the reward
 230 can be a computationally-intensive task (all the more so when it requires solving high-dimensional
 231 discretization of partial differential equation systems), hence the number of state-action-reward triplets
 232 available from the current policy is generally limited. PBO therefore improves the reliability of the loss
 233 evaluation by incorporating the reward data available from several previous generations, using a decay
 234 parameter $\eta \in [0, 1]$ to give recent generations more weight by exponentially decreasing the advantage
 235 history. A rule of thumb for the decay factor used in the remainder of the paper is given by

$$\eta = 1 - e^{-\alpha d}, \quad (9)$$

236 with $\alpha > 0$ to retain a longer memory of the previous individuals as the problem dimensionality d
 237 increases (very much consistent with the idea that more individuals are then needed to build a coherent
 238 covariance matrix). The decrease rate is set empirically to $\alpha = 0.35$, hence $\eta = 0.5$ for $d = 2$, 0.82 for
 239 $\eta = d = 5$, and $\eta = 0.98$ for $d = 10$. In practice, each of the three neural networks are updated for n_e

240 epochs (the number of full passes of the algorithm over the entire data set) using a learning rate λ_r
 241 and a history of n_g generations, shuffled and organized in n_b mini-batches (whose size are in multiples
 242 of n_i , the number of individuals sampled at each generation). An important attribute of PBO is that
 243 all three networks can use different meta-parameters and network architectures, which we show in the
 244 following can substantially impact the convergence rate.

245 3.5 Connection to evolutionary strategies

246 While intrinsically a single-step policy-gradient algorithm, several PBO features are reminiscent of the
 247 CMA-ES algorithm introduced in section 2. The main analogies are as follows:

- 248 • Both CMAES and PBO use a multivariate normal distribution parameterized by a full co-
 249 variance matrix to improve the balance between exploration and exploitation. There lies the
 250 main progress with respect to the previous single-step PPO-1 algorithm, that samples actions
 251 isotropically from scalar covariance matrices (this has been shown to possibly lead to bad
 252 coverage of the parameter space in case of irregular topology of the cost function [19].
- 253 • PBO computes the policy loss (5) using only the positive-advantage actions. This keeps the
 254 policy consistent with the collected experience data, and is reminiscent of the elitist selection
 255 of individuals performed in the CMA-ES update [30],
- 256 • PBO uses history of previous generations to update the network parameters, in the same
 257 way CMA-ES uses an evolution path to add information about correlations across consecutive
 258 generations,
- 259 • PBO exponentially decays the advantage history of older generation, which is also a well-known
 260 feature of CMA-ES, where scaled covariance matrices from past generations are re-used in
 261 future updates and the influence of previous steps decays exponentially in the evolution path
 262 [30].

263 Ultimately, PBO can be thought as an evolution strategy without a specific update rule, in the sense
 264 that CMA-ES relies on internal analytical update rules to directly output the probability density
 265 function parameters, while the update rules of PBO are on the neural network outputs used to design
 266 valid probability density function parameters.

267 4 Minimization of analytic functions

268 4.1 Test cases

269 This section considers simple minimization problems on a set of analytic functions classically exploited
 270 for benchmarking purposes of optimization methods:

- 271 • the two-dimensional (2-D) parabola function, whose global minimum is in (0,0), with a search
 272 domain equal to $[-5, 5]^2$ and a starting point at (2.5, 2.5):

$$f(x_1, x_2) = x_1^2 + x_2^2, \quad (10)$$

- 273 • the d -dimensional (d -D) Rosenbrock function, whose global minimum is in $(1, \dots, 1)$ and stands
 274 in a very narrow valley notoriously difficult to catch for optimization algorithms (three cases
 275 $d = 2, 5$ and 10 are tackled for comparison), with a search domain equal to $[-2, 2]^d$ and a
 276 starting point at $(-1, 0)$ in 2-D, and $(0, \dots, 0)$ in 5-D and 10-D:

$$f(x_1, \dots, x_d) = \sum_{i=1}^{d-1} (1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2, \quad (11)$$

- 277 • the 2-D Branin function, that has two identical global minima at $(\pi, 2.275)$ and $(3\pi, 2.275)$,
 278 with a search domain equal to $[0, 15]^2$ and a starting point at (7.5, 7.5):

$$f(x_1, x_2) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10, \quad (12)$$

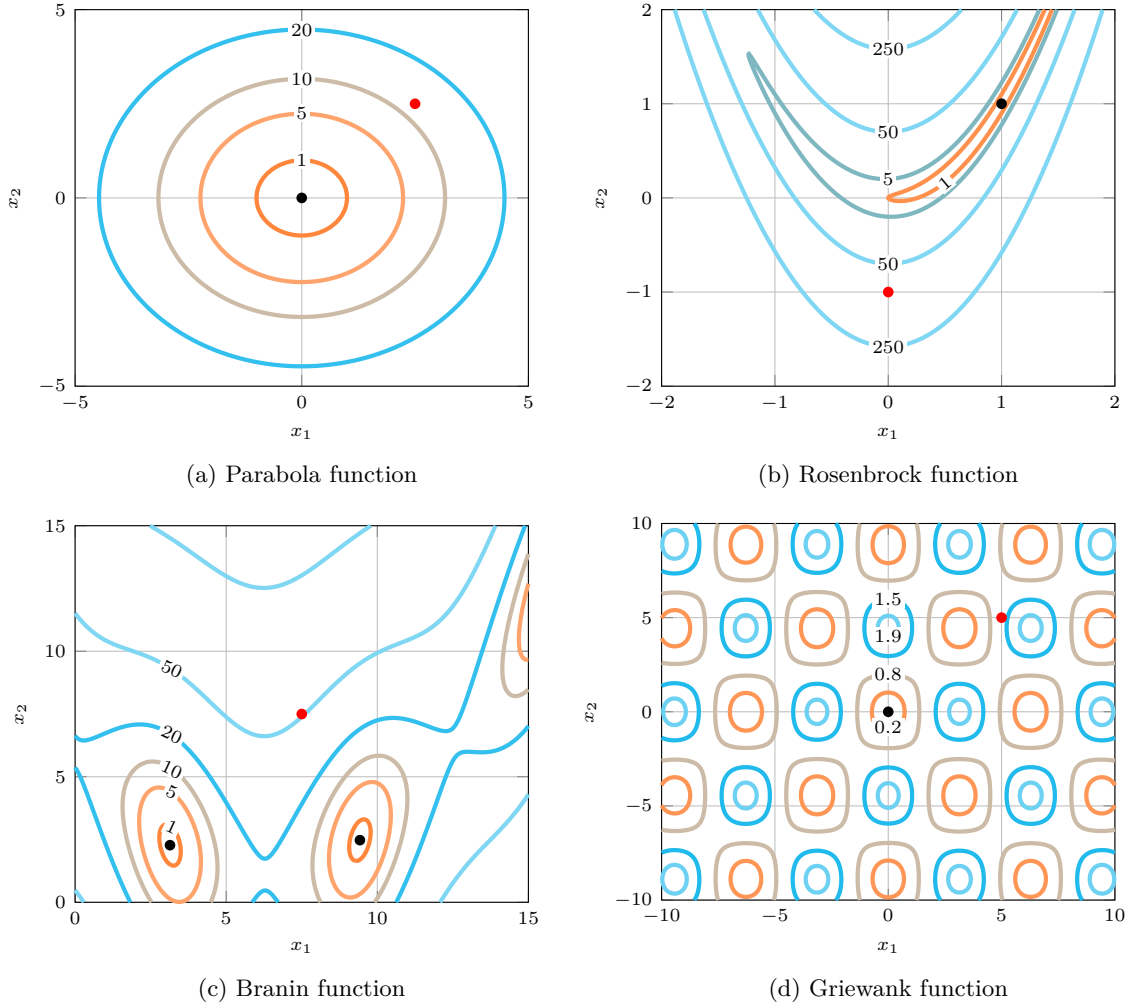


Figure 4: **2D analytic functions used as targets for minimization problems.** The global minima and starting points are reported as the black and red dots, respectively.

- 279 • the 2-D Griewank function, that has multiple widespread, regularly distributed, identical local
 280 minima, and only one global minimum at $(0, 0)$, with a search domain equal to $[-10, 10]^2$ and
 281 a starting point at $(5, 5)$:

$$f(x_1, x_2) = 1 + \frac{x_1^2 + x_2^2}{4000} - \cos(x_1) \cos\left(\frac{x_2}{\sqrt{2}}\right). \quad (13)$$

282 The 2-D functions are presented in figure 4 on their respective domains. In this section, we follow the
 283 CMA-ES rules of thumb and set the number of individuals per generation to:

$$n_i = \lceil 4 + 3 \ln(d) \rceil. \quad (14)$$

284 For each case, PBO is benchmarked against our previous single-step PPO-1 algorithm [18, 19, 20] as
 285 well as (μ, λ) -ES and CMA-ES algorithms implemented in in-house production codes. To ensure a
 286 fair comparison, the initial parameters, number of individuals per generation and starting points are
 287 identical for all methods, as indicated in figure 4. Moreover, a large initial standard deviation is used
 288 by default, to ensure a good exploration of the optimization domain.

	m	σ	ρ
λ_r	5×10^{-3}	5×10^{-3}	1×10^{-3}
n_g	1	8	16
n_e	128	8	8
n_b	1	4	8
Arch.	[2, 2, 2]	[2, 2, 2]	[2, 2, 2]

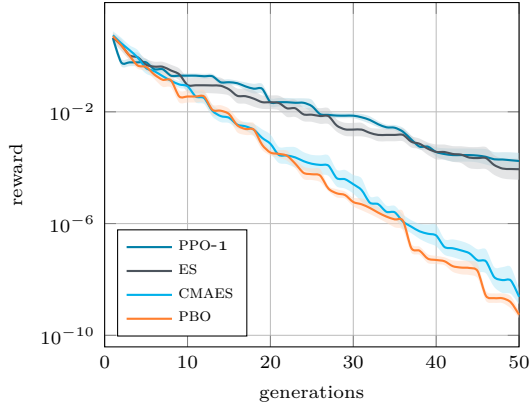
Table 1: **Detail of the networks achitecture and PBO meta-parameters.** As mentioned in section 3.1, λ_r is the learning rate, n_g is the number of generations used for learning, n_e is the number of epochs, and n_b is the number of mini-batches. For the architecture, only the sizes of the hidden layers are given.

289 4.2 Results

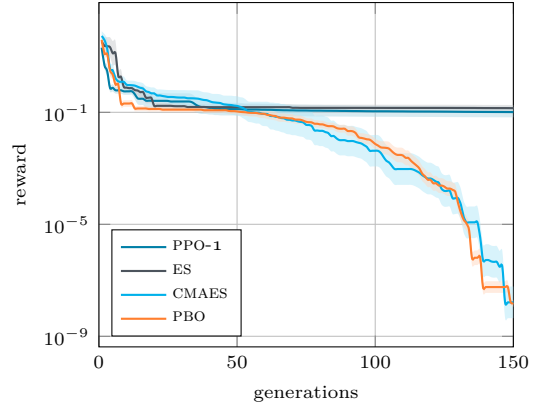
290 In order to emphasize flexibility and generalizability, all benchmarks are tackled without fine-tuning
291 of the algorithm, *i.e.*, all runs use the same PBO meta-parameters listed in table 1, hence the results
292 documented hereafter should be understood as a baseline measure of performance for which there is
293 ample room for improvement. For each considered case, we present in figure 5 the evolution of the best
294 individual cost during the optimization process of a given algorithm. Performances are averaged over 10
295 runs, with standard deviations shown as the light shade around. PBO can be seen to perform extremely
296 well on the parabola, the Branin and the 2-D Rosenbrock functions, as it significantly outperforms
297 PPO-1 and (μ, λ) -ES (both of which perform remarkably similarly) and generally achieves convergence
298 rates and final cost levels similar to CMA-ES. The anisotropy of the PBO optimization process is further
299 illustrated in figure 6 for the 2-D Rosenbrock function: starting in $(0, -1)$ with a large initial variance,
300 the algorithm quickly descends toward the entrance of the narrow valley, in the vicinity of $(0, 0)$. After
301 a few tens of generation for exploration, the algorithm figures out the shape of the valley entrance, the
302 search distribution starts to elongate, progresses into the valley, before reaching the global minimum
303 within approximately 100 generations. PBO performs worst on the Griewank function, as the solutions
304 quickly become trapped by one of the local minima due to the inability to set a suitable step size for
305 the local search process (but all methods considered suffer from the same lack of exploration, and
306 ultimately perform almost identically under the same test conditions). In larger dimensions, PBO
307 shows faster convergence and better performance at intermediate stages (here on the 5-D and 10-D
308 Rosenbrock functions). This experiment confirms the capabilities of PBO to efficiently elongate its
309 research area with respect to the local shape of the cost function, and to converge in moderately large
310 research spaces.

311 We revisit now the 2-D Rosenbrock benchmark and assess the performance sensitivity to the PBO
312 meta-parameters, using the above results (obtained with those meta-parameters listed in table 1) as
313 reference. It can be seen from figure 7a that a larger number of individuals per generation n_i leads
314 to a faster convergence. Such a finding is very much consistent with expectations as it proceeds from
315 both a more accurate evaluation of the loss function (5) and a richer exploration of the search space.
316 Nonetheless, the performance remains surprisingly decent with as little as 3 individuals per generation.
317 The final performance levels seem to saturate around 1×10^{-8} , which we believe is a side-effect of the
318 neural network training process. This is a point that deserves further consideration, although the
319 saturation value is small enough that it likely has little to no effect in practical optimization problems.

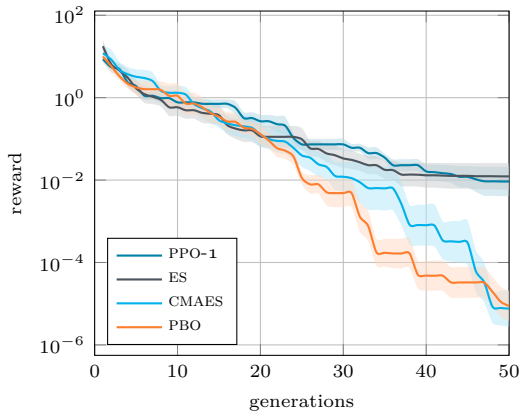
320 The architecture of the neural networks also affects performance in a major way, as we show in figure
321 7b that increasing the networks depth and width can make PBO out-perform its reference benchmark
322 (and thus CMA-ES). This means that PBO does indeed exploit the large network parameter state
323 as a proxy to perform efficient optimization, in contrast to just optimizing the bias of the last layer
324 while keeping all weights to zero. Also, PBO being a stochastic method, using deeper networks also
325 substantially increases the performance stability from one run to another. Yet, beyond a certain point,
326 detrimental effects are observed, which can be attributed to vanishing gradients and/or too large
327 parameters states (not shown here). In the same vein, additional numerical experiments (not shown
328 here) conducted on the 5-D and 10-D Rosenbrock functions suggest that these conclusions do not carry
329 over easily to larger dimensional search spaces, and the reference network architecture used in section



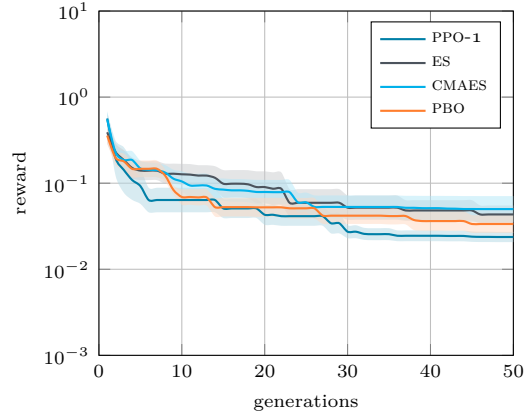
(a) 2-D parabola function



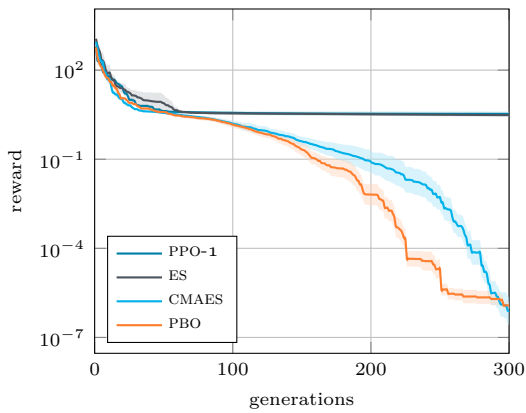
(b) 2-D Rosenbrock function



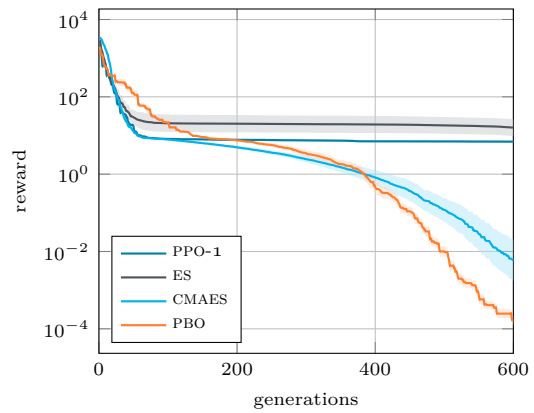
(c) 2-D Branin function



(d) 2-D Griewank function



(e) 5-D Rosenbrock function



(f) 10-D Rosenbrock function

Figure 5: **Minimization problems on analytic functions**, using PBO, PPO-1, ES and CMAES. To ensure a fair comparison, the initial parameters and starting points of the three methods are identical, and the same number of individuals per generation is used for the four methods.

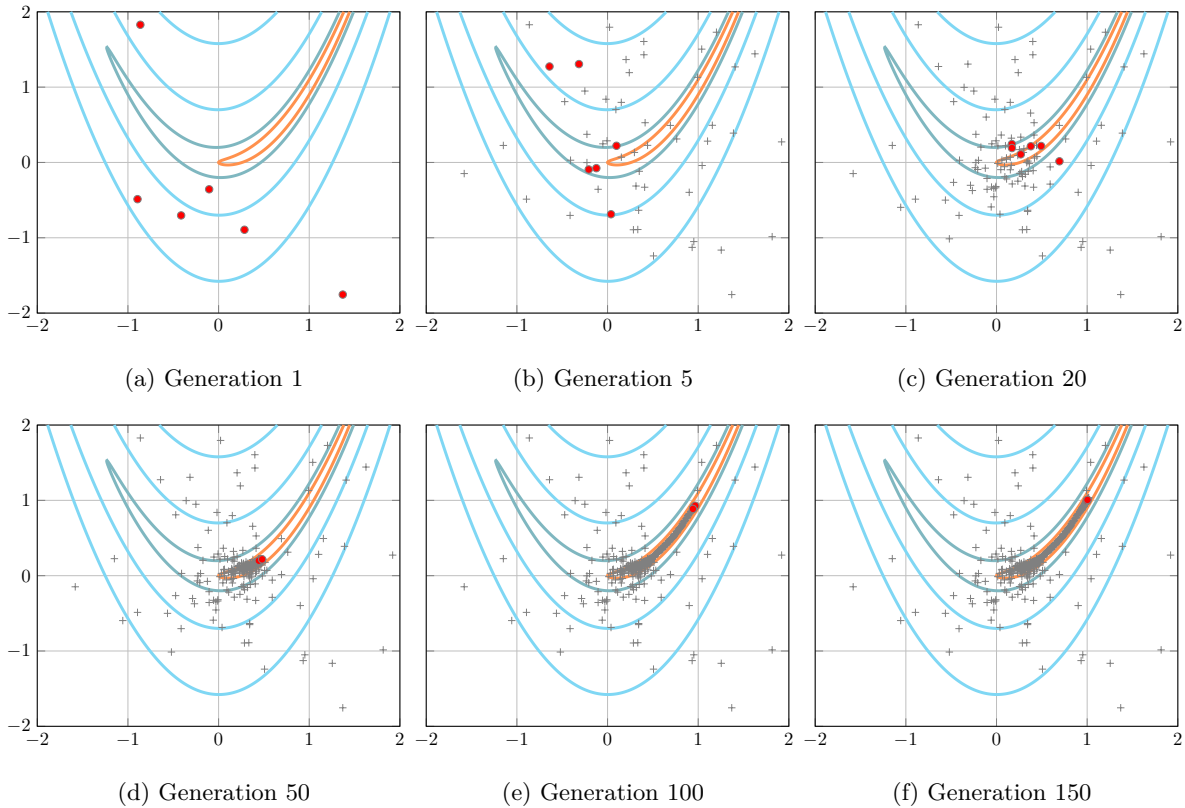


Figure 6: **Successive generations of the PBO algorithm** during a single minimization run of the 2D Rosenbrock function. The red dots indicate the individuals of the current generation, while gray crosses correspond to the individuals of all previous generations.

330 4.2 ends up being a good overall candidate. The general picture to be drawn is that PBO exhibits
 331 strong performance and is very promising for use in more applied optimization problems, but that
 332 further characterization and fine-tuning are mandatory to outperform more advanced methods on a
 333 consistent basis.

334 5 Parametric control laws for the Lorenz attractor

335 This section considers the optimization of parametric control laws for the Lorenz attractor, a simple
 336 nonlinear dynamical system representative of thermal convection in a two-dimensional cell [36]. The
 337 set of governing ordinary differential equations reads:

$$\begin{aligned}
 \dot{x} &= \sigma(y - x), \\
 \dot{y} &= x(\rho - z) - y, \\
 \dot{z} &= xy - \beta z,
 \end{aligned}
 \tag{15}$$

338 where σ is related to the Prandtl number, ρ is a ratio of Rayleigh numbers, and β is a geometric
 339 factor³. Depending on the values of the triplet (σ, ρ, β) , the solutions to (15) may exhibit chaotic
 340 behavior, meaning that arbitrarily close initial conditions can lead to significantly different trajectories
 341 [37], one common such triplet being $(\sigma, \rho, \beta) = (10, 28, 8/3)$, that leads to the well-known butterfly
 342 shape presented in figure 8. A parametric control law is introduced in the following to alleviate or curb

³The ρ and σ used here are therefore the canonical notations of the Lorenz attractor parameters, and have no link with the standard deviations and correlation parameters used previously in the paper.

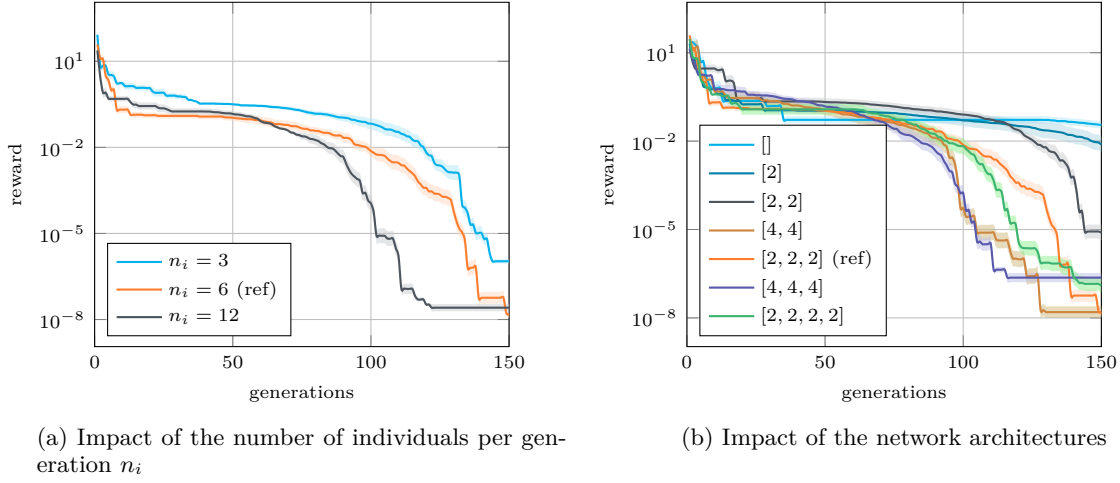


Figure 7: **Sensitivity of the PBO convergence properties to the number of individuals per generation and network architectures.** The reference solutions obtained using the parameters listed in table 1 and shown in figure 5 are reproduced as the orange curves.

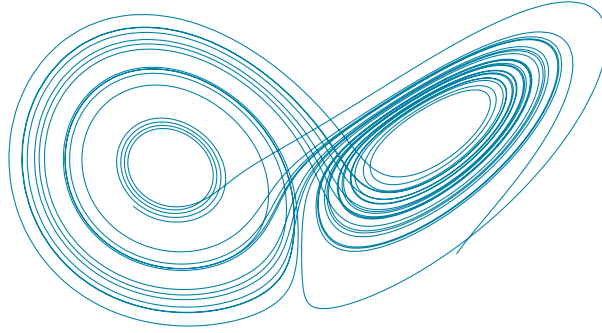


Figure 8: **Chaotic sampled solution of the Lorenz attractor**, computed by time-integration of (15) with $(\sigma, \rho, \beta) = (10, 28, 8/3)$, from initial conditions $(x_0, y_0, z_0) = (10, 10, 10)$ over 30 time units. The presented view is in the $x-z$ phase plane.

343 such chaotic behavior, whose design parameters are optimized by PBO with respect each intended
 344 control objective.

345 5.1 Parametric control law

346 We build here on existing control attempts of the Lorenz system [38] and add to (15) a feedback control
 347 on the y variable for the controlled system to be:

$$\begin{aligned} \dot{x} &= \sigma(y - x), \\ \dot{y} &= x(\rho - z) - y + u(\dot{x}, \dot{y}, \dot{z}), \\ \dot{z} &= xy - \beta z, \end{aligned} \quad (16)$$

348 where u is the feedback velocity defined as:

$$u(\dot{x}, \dot{y}, \dot{z}) = \tanh(w_x \dot{x} + w_y \dot{y} + w_z \dot{z} + b), \quad (17)$$

349 in a way such that $|u| < 1$, and w_x, w_y, w_z and b are the true free parameters to optimized (hence
 350 $d = 4$). The control law (17) is meant to mimic the output of an artificial neuron, with w_x, w_y and

351 w_z being the weights of the neuron inputs, and b representing its bias. We set the initial condition to
 352 $(x_0, y_0, z_0) = (10, 10, 10)$, and the attractor parameters to $(\sigma, \rho, \beta) = (10, 28, 8/3)$ for the uncontrolled
 353 system to be chaotic. In practice, we use scaled inputs:

$$(\hat{x}, \hat{y}, \hat{z}) = \left(\frac{\dot{x}}{x_s}, \frac{\dot{y}}{y_s}, \frac{\dot{z}}{z_s} \right), \quad (18)$$

354 using scaling factors $(x_s, y_s, z_s) = (15, 20, 40)$ representative of the approximate maximal amplitude
 355 reached by each variable of the uncontrolled problem, which allows seeking all optimal parameters w_x^* ,
 356 w_y^* , w_z^* , and b^* in $[-1, 1]$ (as required by PBO). In the following, we solve system (16) using the `odeint`
 357 function of the Scipy package [39]. The system is evolved control-free for 5 time units (from $t = -5$
 358 to $t = 0$), after which the control kicks in for 25 time units, from $t = 0$ to $t = 25$. The integration
 359 time-step is fixed, and set to $\Delta t = 0.01$ time units. All considered cases are tackled with the same
 360 reference meta-parameters listed in table 1 (again to highlight the robustness and versatility of the
 361 method before aiming to fine-tune the performance), only the number of individuals per generation n_i
 362 is raised to 16 due to the chaotic nature of the system and the limited computational cost required to
 363 integrate the problem.

364 5.2 Lorenz stabilizer

365 Small control actuation on the \dot{y} evolution equation is first use to stabilize the Lorenz system in the
 366 $x < 0$ quadrant (as is done in [38]) using the reward function:

$$r = \Delta t \sum_{i=0}^{n_t} \{x_i < 0\}, \quad (19)$$

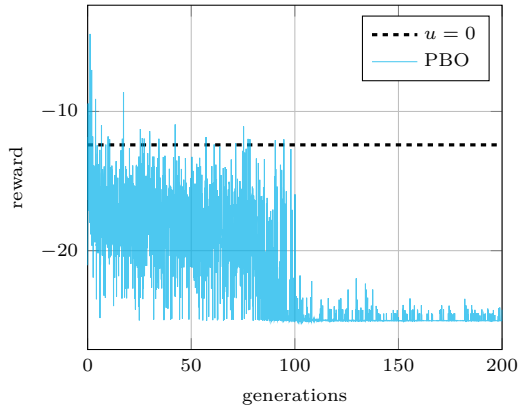
367 where n_t is the total number of time-steps, and $\{x_i < 0\} = 1$ if $x_i < 0$, and 0 otherwise, in a way such
 368 that the reward is large if and only if the x coordinate remains within the targeted domain. The reward
 369 function is multiplied by Δt with the only purpose to make it independent of the time discretization.
 370 For the sake of clarity, the results presented in figure 9a pertain to a single run (not an average over
 371 runs,) which is because the chaotic behavior of the attractor yields a significantly distorted reward
 372 history. Even though, the PBO algorithm converges after approximately 100 generations (with good
 373 reward values are obtained after a few ten generations). The subsequent variations are ascribed to the
 374 reward function. Indeed, it is flat by design for any value $x < 0$, and therefore does not promote sharp
 375 convergence to a specific value of x , although we show in figure 9b that all four control parameters
 376 converge to well-defined, non-trivial values. The efficiency of the control is further illustrated in figure
 377 9c showing that optimally controlled attractor is successfully confined in the $x < 0$ basin just 5 time
 378 units after the control has kicked in.

379 5.3 Lorenz oscillator

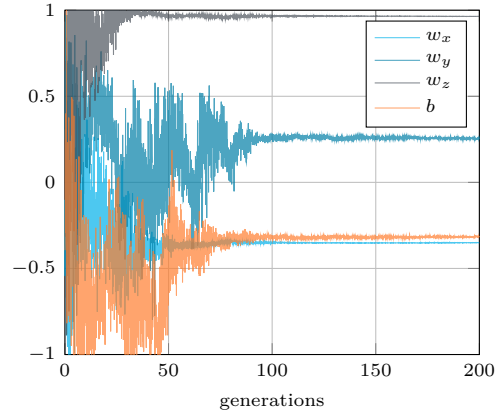
380 Similar small control actuation on the \dot{y} evolution equation is now used to maximize the number of sign
 381 changes of the Lorenz system, as proposed in [38]. This is done using the following reward function:

$$r = \sum_{i=0}^{n_t-1} \{x_i x_{i+1} < 0\}, \quad (20)$$

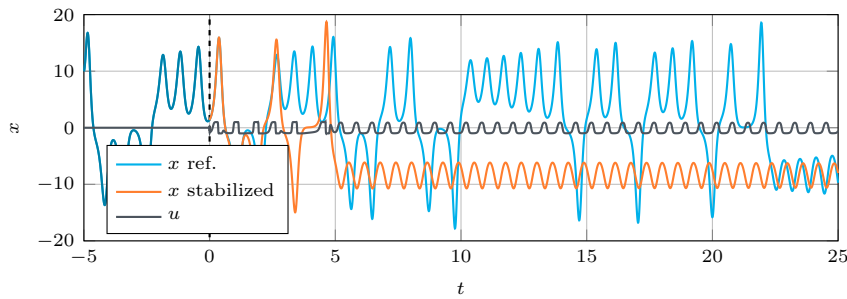
382 in a way such that the reward is large if and only if the x coordinate changes sign in consecutive time
 383 steps. This function is much harder to maximize than its stabilizer counterpart (19) due to its higher
 384 sparsity, *i.e.* the larger proportions of actions yielding a zero instantaneous reward. Such sparsity is
 385 the reason why no sharp convergence is found over the course of a single optimization run, as shown
 386 in figure 10, but there is a clear diminishing trend and the optimally controlled attractor ultimately
 387 exhibits the expected behavior, as it is mostly confined on a narrow orbit that allows it to quickly
 388 oscillate between the $x < 0$ and the $x > 0$ regions.



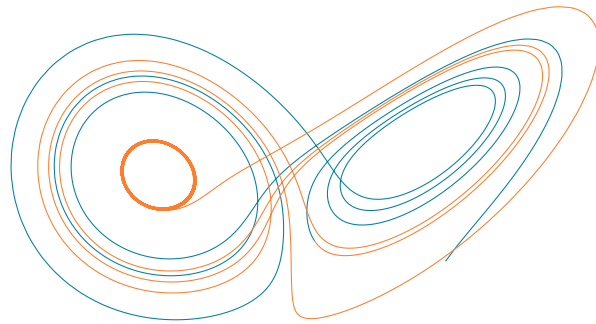
(a) Evolution of the reward function of the Lorenz stabilizer case over a single run. The dashed line indicates the control-free reward level



(b) Evolution of the four control parameters over a single run

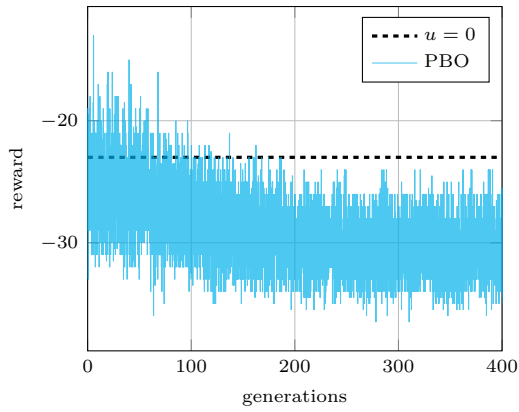


(c) Evolution of the x component with and without optimal parametric control

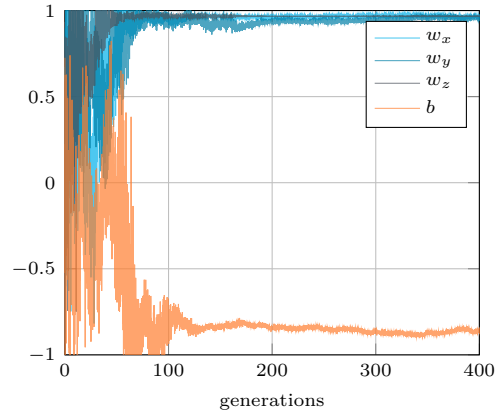


(d) Plot of the Lorenz attractor with optimized stabilizer control, seen in the $x - z$ plane

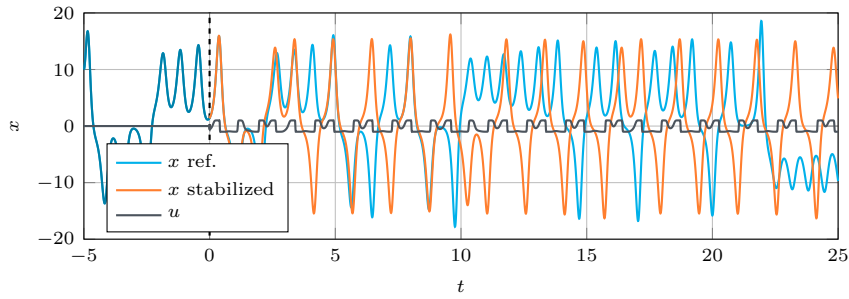
Figure 9: **Results for the Lorenz attractor with optimized stabilizer control.** Given the chaotic nature of the problem, results are provided for a single run only.



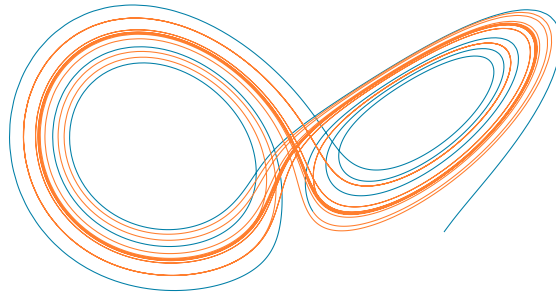
(a) Evolution of the reward function of the Lorenz stabilizer case over a single run. The dashed line indicates the control-free reward level



(b) Evolution of the four control parameters over a single run



(c) Evolution of the x component with and without optimal parametric control



(d) Plot of the Lorenz attractor with optimized oscillator control

Figure 10: Same as figure 9 for the Lorenz oscillator optimization.

389 6 Conclusion

390 This research formally introduces policy-based optimization (PBO), a novel black-box algorithm for
391 optimization and open-loop control problems, at the crossroad of policy gradient methods and evolution
392 strategies. PBO is single-step, meaning that the DRL agent gets only one attempt per learning episode
393 at finding the optimal. It evolves a multivariate normal search distribution whose parameters (including
394 especially a full covariance matrix) are learnable from neural network outputs. The method represents
395 significant improvement with respect to our previous single-step PPO-1 algorithm, that samples actions
396 isotropically from a scalar standard deviation (which can be detrimental when the topology of the
397 cost function is distorted). PBO is shown to outperform classical isotropic ES techniques on the
398 minimization problem of reference analytic functions, up to 10 dimensions. The performance is actually
399 similar to that of CMA-ES, with moderate advantage on the convergence rate obtained in intermediate
400 dimensions, although PBO is found capable to out-perform its reference benchmark (and thus CMA-
401 ES) by adequate fine tuning of the network architecture. PBO is also applied to the optimization of
402 parametric control law for the Lorenz attractor, for which it successfully stabilizes the Lorenz system
403 in a given domain, or conversely enhances the ability of the system to change sign.

404 Researchers have just begun to gauge the relevance of DRL techniques to assist the design of optimal
405 control strategies. This research weighs in on this issue and shows that PBO holds a high potential
406 as a reliable, go-to black-box optimizer inheriting from both policy gradients and evolutionary strat-
407 egy methods. In this respect, the method can thus benefit from the solid background acquired in
408 evolutionary computations, and from rapid progresses achieved by the DRL community. Despite the
409 present achievements, further development, characterization and fine-tuning are needed to consolidate
410 the acquired knowledge: multiple refinements can be considered, including extending the scope to
411 deterministic policy gradient techniques [40], or using importance sampling weights [41] to replace
412 the exponential decay heuristic herein proposed. We certainly welcome such initiatives and purposely
413 make the source code available upon request via a dedicated Github repository [42].

414 Acknowledgements

415 This work is supported by the Carnot M.I.N.E.S. Institute through the M.I.N.D.S. project.

416 References

- 417 [1] W. Rawat and Z. Wang. Deep convolutional neural networks for image classification: a compre-
418 hensive review. *Neural Computation*, 29:2352–2449, 2017.
- 419 [2] A. Khan, A. Sohail, U. Zahoor, and A. S. Qureshi. A survey of the recent architectures of deep
420 convolutional neural networks. *Artificial Intelligence Review*, pages 2352–2449, 2020.
- 421 [3] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan. Speech recognition using deep neural
422 networks: a systematic review. *IEEE Access*, 7:19143–19165, 2019.
- 423 [4] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye. A review on generative adversarial networks: algo-
424 rithms, theory, and applications. <http://arxiv.org/abs/2001.06937>, 2020.
- 425 [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller.
426 Playing Atari with deep reinforcement learning. <http://arxiv.org/abs/1312.5602>, 2013.
- 427 [6] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker,
428 M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and
429 D. Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550, 2017.
- 430 [7] OpenAI. OpenAI Five. <https://blog.openai.com/openai-five/>, 2018.
- 431 [8] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic
432 for image-based robot learning. <http://arxiv.org/abs/1710.06542>, 2017.
- 433 [9] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An
434 actor-critic algorithm for sequence prediction. <http://arxiv.org/abs/1607.07086>, 2016.
- 435 [10] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and
436 A. Shah. Learning to drive in a day. <http://arxiv.org/abs/1807.00412>, 2018.

- 437 [11] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall. Learning to drive
438 from simulation without real world labels. <http://arxiv.org/abs/1812.03823>, 2018.
- 439 [12] W. Knight. Google just gave control over data center cool-
440 ing to an AI. [http://www.technologyreview.com/s/611902/
441 google-just-gave-control-over-data-center-cooling-to-an-ai/](http://www.technologyreview.com/s/611902/google-just-gave-control-over-data-center-cooling-to-an-ai/), 2018.
- 442 [13] G. Villarrubia, J. F. De Paz, P. Chamoso, and F. De la Prieta. Artificial neural networks used in
443 optimization problems. *Neurocomputing*, 272:10–16, 2018.
- 444 [14] A. M. Schweidtmann and A. Mitsos. Deterministic global optimization with artificial neural
445 networks embedded. *Journal of Optimization Theory and Applications*, 180:925–948, 2019.
- 446 [15] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shilling-
447 ford, and N. de Freitas. Learning to learn by gradient descent by gradient descent.
448 <http://arxiv.org/abs/1606.04474>, 2016.
- 449 [16] X. Yan, J. Zhu, M. Kuang, and X. Wang. Aerodynamic shape optimization using a novel optimizer
450 based on machine learning techniques. *Aerospace Science and Technology*, 86:826–835, 2019.
- 451 [17] R. Li, Y. Zhang, and H. Chen. Learning the aerodynamic design of supercritical airfoils through
452 deep reinforcement learning. <https://arxiv.org/abs/2010.03651>, 2020.
- 453 [18] J. Viquerat, J. Rabault, A. Kuhnle, H. Ghraieb, A. Larcher, and E. Hachem. Direct shape
454 optimization through deep reinforcement learning. *Journal of Computational Physics*, 428:110080,
455 2021.
- 456 [19] H. Ghraieb, J. Viquerat, A. Larcher, P. Meliga, and E. Hachem. Optimization and passive flow
457 control using single-step deep reinforcement learning. <http://arxiv.org/abs/2006.02979>, 2020.
- 458 [20] E. Hachem, H. Ghraieb, J. Viquerat, A. Larcher, and P. Meliga. Deep reinforcement
459 learning for the control of conjugate heat transfer with application to workpiece cooling.
460 <https://arxiv.org/abs/2011.15035>, 2020.
- 461 [21] P. Hämaläinen, A. Babadi, X. Ma, and J. Lehtinen. Ppo-cma: Proximal policy optimization with
462 covariance matrix adaptation. <http://arxiv.org/abs/1810.02541>, 2018.
- 463 [22] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal ap-
464 proximators. *Neural Networks*, 2(5):359–366, 1989.
- 465 [23] I. Goodfellow, Y. Bengio, and A. Courville. *The Deep Learning Book*. MIT Press, 2017.
- 466 [24] R. Sutton, D. Mcallester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement
467 learning with function approximation. *Adv. Neural Inf. Process. Syst.*, 12, 2000.
- 468 [25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating
469 errors. *Nature*, 323:533–536, 1986.
- 470 [26] V. R. Konda and J. N. Tsitsiklis. Actor-critic algorithms. In *Advances in neural information
471 processing systems*, pages 1008–1014, 2000.
- 472 [27] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control
473 using generalized advantage estimation. <https://arxiv.org/abs/1506.02438>, 2015.
- 474 [28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization
475 algorithms. <http://arxiv.org/abs/1707.06347>, 2017.
- 476 [29] R. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- 477 [30] N. Hansen. The cma evolution strategy: A tutorial. <http://arxiv.org/abs/1604.00772>, 2016.
- 478 [31] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization.
479 <http://arxiv.org/abs/1412.6980>, 2014.
- 480 [32] R. Rebonato and P. Jäckel. The most general methodology to create a valid correlation matrix
481 for risk management and option pricing purposes. *Available at SSRN 1969689*, 2011.
- 482 [33] F. Rapisarda, D. Brigo, and F. Mercurio. Parameterizing correlations: a geometric interpretation.
483 *IMA Journal of Management Mathematics*, 18(1):55–73, 2007.
- 484 [34] K. Numpacharoen and A. Atsawarungrangkit. Generating correlation matrices based on the
485 boundaries of their coefficients. *PLOS One*, 7(11), 2012.

- 486 [35] S. Maree. Correcting non positive definite correlation matrices. BSc Thesis Applied Mathematics,
487 TU Delft, 2012.
- 488 [36] B. Saltzman. Finite amplitude free convection as an initial value problem. Journal of atmospheric
489 sciences, 19(4):329–341, 1962.
- 490 [37] E. N. Lorenz. Deterministic nonperiodic flow. Journal of Atmospheric Sciences, 20(2):130–141,
491 1963.
- 492 [38] G. Beintema, A. Corbetta, L. Biferale, and F. Toschi. Controlling rayleigh–bénard convection via
493 reinforcement learning. Journal of Turbulence, 21(9-10):585–605, 2020.
- 494 [39] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski,
495 P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman,
496 N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng,
497 E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero,
498 C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, and P. van Mulbregt. SciPy 1.0:
499 Fundamental algorithms for scientific computing in python. Nature Methods, 17:261–272, 2020.
- 500 [40] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra.
501 Continuous control with deep reinforcement learning. <https://arxiv.org/abs/1509.02971v6>, 2019.
- 502 [41] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample
503 efficient actor-critic with experience replay. <https://arxiv.org/abs/1611.01224>, 2017.
- 504 [42] J. Viquerat. PBO git repository. <https://github.com/jviquerat/pbo>, 2021.