Learning Global Optimization by Deep Reinforcement Learning

Moésio W. da Silva Filho ✉[1], Gabriel A. Barbosa[1], and Péricles B. C. Miranda[1]

Federal Rural University of Pernambuco, Recife, Pernambuco 52171-900, Brazil
{moesio.wenceslau,gabriel.augusto,pericles.miranda}@ufrpe.br

Abstract. Learning to Optimize (L2O) is a growing field that employs a variety of machine learning (ML) methods to learn optimization algorithms automatically from data instead of developing hand-engineered algorithms that usually require hyperparameter tuning and problem-specific design. However, there are some barriers to adopting those learned optimizers in practice. For instance, they exhibit instability during training, poor generalization to problems outside the distribution, and lack scalability. Current research efforts suggest either improving L2O models or improving training techniques to overcome such hardships. We focus on the latter and propose to train a Deep Reinforcement Learning (Deep RL) agent to learn an optimization algorithm from training in a diverse set of benchmark functions. To this end, we propose a general framework for learning to optimize by reinforcement learning, which adapts training strategies used in other L2O approaches, such as curriculum learning and input normalization. We investigate the importance of these strategies through an ablation study and show that even though Deep RL, to the best of our knowledge, is not a well-explored theme in L2O, it provides a direct framework to learn an optimizer able to deal with the exploration-exploitation dilemma and that the applied techniques improved stability and generalization.

Keywords: Learning to Optimize · L2O · Deep Reinforcement Learning.

1 Introduction

Many problems in Computer Science can be formalized as optimization problems [18], in which the best solution, for some criteria, must be selected from a set of possible solutions. Thus, many optimization algorithms have been developed over the years. However, the manual design of optimization algorithms is a laborious process [4] that usually requires domain-specific knowledge and human experts [13].

Learning to Optimize (L2O) [1, 4–7, 13–16, 19, 21] is a rising field of machine learning (ML) that aims to replace hand-designed optimization algorithms with learned optimizers. Usually, L2O methods learn a parameterized model that acts as an *optimizer* for a series of similar optimization problems called *optimizees* [5].

It is fairly common that the optimizees are learning problems [5], in which case, L2O can be viewed from a meta-learning perspective [1]. However, recent work considered other optimization problems that are not directly related to learning, such as benchmark functions [7], and protein docking [4].

The mainstream in L2O leverages recurrent neural networks (RNNs), typically long-short term memory (LSTM), as the model for the optimizer [1,4,14,21]. However, there are some barriers to adopting those learned optimizers in practice. For instance, training those optimizers is difficult [16], and they suffer from poor generalization [5]. Thus, the current effort suggests either improving L2O models, such as using hierarchical RNNs [21], or improving training [5, 16].

We focus on the latter and train a Deep Reinforcement Learning (deep RL) agent to learn global optimization. Previous work with RL in L2O have considered slightly different formulations. For instance, in [7] the RL agent learned a component of the optimizer, while in [13] the agent was the optimizer. We argue that there is a need for a concise and flexible framework in applying Deep RL for learning to optimize, enabling us to use advances in other fields of RL, such as Multi-Task Reinforcement Learning and Meta Reinforcement Learning. To this end, we propose a general framework for learning to optimize by Deep Reinforcement Learning.

Our Contributions We offer a framework for representing L2O problems from a Reinforcement Learning perspective, taking into account previous work and merging proposed strategies for improving training and stability of other learnable optimizers. The proposed framework is flexible and resembles the multi-task RL setting. We investigate the proposed framework to learn global optimization from training in set of 26 benchmark functions and obtain good preliminary results for 2-dimensional problems.

2 Preliminaries

In this section, we cover the basic concepts used throughout this paper. We review Deep Reinforcement Learning, POMDPs and Learning to Optimize (L2O).

2.1 Deep Reinforcement Learning

Deep Reinforcement Learning is a field that combines Reinforcement Learning (RL), which deals with sequential decision-making through an agent that takes actions in an environment, and Deep Learning, which employs Deep Neural Networks, enabling RL to scale to problems with high-dimensional state, and action spaces [2].

General RL problems can be defined as discrete-time stochastic control processes where an *agent* interacts with an *environment* to learn a behavior that optimizes a notion of *cumulative rewards*. The formal framework for the process depends on the task at hand. Markov Decision Processes (MDPs) and Partially Observable MDPs (POMDPs) are common approaches [2]. In the following paragraphs, we define POMDPs.

POMDPs are particularly useful for real-world applications since many decision-making problems are partially observable by nature, and issues with sensors (e. g., noise, data loss, occlusions) may limit the agent's perceptual abilities [11]. Another source of partial observability comes from the environment design, which might not fully capture the underlying states.

A POMDP is a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{O}, F, U, R, b_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{O}$ is the observation space, $F(s_{t+1}|s_t, a_t)$ is the transition function (also called *dynamics* or *model*), $U(o_{t+1}|s_{t+1})$ is the emission function, $R(s_t, a_t, s_{t+1})$ is the reward function, and b_0 is the distribution over initial states s_0.

Under a POMDP, the agent-environment interaction is as follows: for every time step $t \in \mathbb{N}$, the environment is in a state $s_t \in \mathcal{S}$, the agent receives an observation $o_t \sim O(o_t|s_t)$ and takes an action $a_t \in \mathcal{A}$. Then, the environment transitions to a new state $s_{t+1} \sim F(s_{t+1}|s_t, a_t)$ and the agent receives a reward $r_{t+1} = R(s_t, a_t, s_{t+1})$.

In this formulation, the goal of the agent [11] is to learn a policy $\pi(a_t|o_{\leq t}, a_{\leq t})$, where $o_{\leq t} = (o_0, \ldots, o_t)$ and $a_{\leq t} = (a_0, \ldots, a_{t-1})$, that maximizes the expected return

$$\bar{R} = \mathbb{E}_{p(\tau)} \left[\sum_{t=1}^{T} \gamma^{t-1} r_t \right], \tag{1}$$

over trajectories $\tau = (s_0, a_0, \ldots, a_{T_1}, s_T)$ induced by π, where $0 \leq \gamma \leq 1$ is the discount factor, and T is the trajectory length which may vary between episodes.

Additionally, as in [17], we refer to s_t^o as the observable state and s_t^h as the hidden state. The observable state is the portion of s_t that can be directly unveiled by the observation o_t, while the hidden state is the remainder of s_t.

2.2 Problem Setup for L2O

In its more general form, L2O deals with the problem of learning an optimizer $\mathcal{U}(\cdot;\theta)$ parameterized by θ that is capable of optimizing functions $f(\mathbf{x})$, called *optimizees*, of a similar class [5]. To this end, $\mathcal{U}(\cdot;\theta)$ is trained to minimize an objective function $\mathcal{L}(\theta)$ over a distribution of optimizees $\mathcal{F}_{\text{train}}$. Then, the learned optimizer is used to iteratively update the optimizees' variables: $\mathbf{x}_{t+1} = \mathbf{x}_t + \mathcal{U}(\cdot;\theta)$.

However, this general form does not apply to all L2O approaches. For instance, [3] learned an optimizer by searching for a mathematical rule. For the sake of simplicity, we will consider the previous problem setup since it applies to many works in the field.

Table 1 summarizes different L2O approaches by comparing the choices of inputs features, optimizer architecture, and training/evaluation set of optimizees.

2.3 Strategies for improving L2O Training

Most of the works in L2O propose strategies, techniques and tricks to improve learning, stability and generalization of learned optimizers by either improving the optimizer model [4, 21] or improving training [6, 15, 16].

Table 1. Summary and comparison of different L2O approaches.

Reference	Architectures	Inputs	Optimizees
[1]	LSTM	$\nabla f(\mathbf{x})$	Quadratic functions; Neural Networks;
[6]	LSTM; DNC	$f(\mathbf{x})$; $\mathbf{x}$	GPs; Benchmarks problems
[19]	LSTM	$f(\mathbf{x})$; $\mathbf{x}$	GMM; Benchmarks problems
[15]	LSTM	$\nabla f(\mathbf{x})$	Neural Networks
[21]	Hierarchical RNN	$\boldsymbol{g}$; γ; η	Benchmark problems; Neural Networks
[4]	Population of LSTMs	$\nabla f(\mathbf{x})$; $\boldsymbol{m}$; $\boldsymbol{v}$; $\boldsymbol{a}$;	Benchmark problems; Protein Docking

$\boldsymbol{g}$: scaled averaged gradients; $\boldsymbol{m}$: gradient momentum;
γ: relative log gradient magnitudes; $\boldsymbol{v}$: particle velocity;
η: relative log learning rate; $\boldsymbol{a}$: particle attraction;

In the next paragraphs, we list some of the proposed strategies for improving the training of L2O approaches and discuss how they were implemented in different L2O RNN-based frameworks.

Augmented training set. One of the proposed techniques consists in augmenting the training set, that is, the optimizees distribution $\mathcal{F}_{\text{train}}$. [21] points out that previously learned optimizers have failed to generalize for functions outside $\mathcal{F}_{\text{train}}$, and a possibility to address this issue is to include a richer set of training functions that better capture the properties of commonly encountered loss landscapes. [5] suggests a similar approach; it is argued that a good optimizer needs to explore and experience various optimizees landscapes in order to generalize well.

Specifically, [21] proposed the $\mathcal{F}_{\text{train}}$ to have optimizees of five classes: (i) benchmark functions (e. g., Ackley, Rosenbrock, Beale, Booth); (ii) well-behaved convex problems (e. g., quadratic functions, logistic regressions); (iii) problems with noisy gradients and minibatches; (iv) slow convergence problems (e. g., oscillating valleys); and (v) transformed problems. Instead, [5] proposed to increase the number of sampled optimizees $f \sim \mathcal{F}_{\text{train}}$, and to re-use sampled optimizees but starting $\mathbf{x}_0$ at different locations.

Input features inspired on optimization algorithms. The choice of input features for a learnable optimizer is crucial for generalization and scalability [16]. [21] argues that choosing features inspired by other optimization algorithms is a way of incorporating useful knowledge in the optimizer. Thus, many works in L2O choose features related to other optimization algorithms.

For instance, [1] considered the gradients $\nabla f(\mathbf{x}_t)$ as input for the optimizer, taking inspiration from the *gradient descent* (GD) algorithm. [21] took inspiration from multiple optimization techniques (e. g., Nesterov momentum) and algorithms (e. g., RMSProp, ADAM), passing the scaled averaged gradients, the relative log gradient magnitudes, and the relative log learning rate as input for the optimizer. On the other hand, [4] proposed to consider features from

Table 2. Common loss functions for learnable optimizers.

Reference	No.	$\mathcal{L}(\theta)$
[1, 6]	1	$\mathbb{E}_f\left[f(\mathbf{x}^*)\right]$
[1, 6]	2	$\mathbb{E}_f\left[\sum_{t=1}^{T}\omega_t f(\mathbf{x}_t)\right]$
[6]	4	$-\mathbb{E}_{f,y_{1:T-1}}\left[\sum_{t=1}^{T}\mathrm{EI}(\mathbf{x}_t\|y_{1:t-1})\right]$
[6]	5	$\mathbb{E}_{f,y_{1:T-1}}\left[\sum_{t=1}^{T}\min\left\{f(\mathbf{x}_t)-\min_{i<t}(f(\mathbf{x}_i)),0\right\}\right]$
[21]	6	$\mathbb{E}_f\left[\frac{1}{T}\sum_{t=1}^{T}(\log(f(\mathbf{x}_t)+\epsilon)-\log(f(\mathbf{x}_0)+\epsilon))\right]$

f: optimizee sampled from $\mathcal{F}_{\text{train}}$; $\mathbf{x}^*$: final optimizee parameters;
ω_t: weight associated with time-step t; ϵ: constant value;
$\mathrm{EI}(\cdot)$: expected posterior improvement of $\mathbf{x}_t$ given observations up to t;

both point-based optimization (gradient and momentum) and population-based optimization (velocity and attraction).

Input normalization. Another proposed technique for improving training stability and generalization of L2O is to apply some sort of input normalization. This can be achieved by either preprocessing inputs or using input features already normalized (e. g., scaled, averaged).

For instance, [1] and [19] used preprocessing functions to normalize the inputs. [1] proposed a gradient preprocessing formula using log and sign. Instead, [19] proposed a preprocessing formula to normalize inputs by considering the mean and variance of previously observed inputs. [21] and [15] used input features that are already normalized and did not need preprocessing.

Improved loss (objective) function for the learnable optimizer. The objective function $\mathcal{L}(\theta)$ is crucial for learning a good optimizer. As such, many objective functions for the learnable optimizer have been proposed. Table 2 summarizes choices in the literature.

Curriculum learning. A novel proposed strategy to further improve training is to use curriculum learning in L2O. [5] proposed to gradually unroll the optimizer more w.r.t the number of epochs. [5] argues that following a curriculum scheme enables the optimizer to start learning short-horizon trajectory patterns and gradually growing to recognize longer dependencies.

Imitation learning. Another novel strategy is to apply imitation learning in L2O. [5] proposed to use analytical optimizers (e. g., ADAM) to stabilize training, prevent overfitting, and improve generalization.

2.4 Global Optimization

An unconstrained global optimization problem can be defined as:

$$\min F(\mathbf{x})$$

$$\text{subject to } L \leq x_i \leq U,\ i = 1, \ldots, D$$

where $F(\mathbf{x})$ is the objective function, $\mathbf{x} = [x_1, \ldots, x_D]^T$ is the vector of decision variables, L and U are the lower and upper bound, respectively.

However, such problems are usually noisy, multi-modal, non-convex, and non-separable, requiring methods to effectively explore the search space [18]. Thus, many evolutionary algorithms are employed. Another approach for solving global optimization problems is to learn to globally optimize [24].

3 Related Work

Initial works in learning to optimize are closely related to *meta-learning*, or *learning to learn* [1, 6, 13–16], which is the idea of acquiring knowledge or inductive bias to accelerate learning. In this sense, most of the *optimization tasks* used to learn an optimizer were learning tasks, such as learning the optimal weights for neural nets [1] and learning to solve control problems [6].

However, recent works have broadened the scope and started to consider optimization tasks that are not directly related to learning, such as benchmark functions [7, 8, 21], and other synthetic functions [24]. Still, learning tasks share similarities to many artificial landscapes, such as being non-convex, multi-modal, and noisy. In the following subsections, we briefly review some related work to our approach.

3.1 Learning to Optimize

Andrychowicz et al. [1] leverage an LSTM as a coordinate-wise optimizer, which takes the preprocessed gradients as input and outputs an update for the optimizee's variables. Chen et al. [6] also consider an LSTM, however using the optimizee's objective values and optimizee's variables as input. Wichrowska et al. [21] introduced a hierarchical RNN architecture for the optimizer. Cao et al. [4] proposed to further expand L2O by considering an algorithmic space of both point-based and population-based optimization algorithms. The authors in [5, 15, 16, 19] proposed a wide range of techniques and tricks for improving learnable optimizers.

3.2 Learning to Optimize by Deep RL

The work proposed by Williams and Peng [22] was one of the firsts to consider RL for function optimization. The proposed approach consisted in training a neural network, through reinforcement learning, to generate better query points for the optimizee. Li and Malik [13] proposed to learn an optimization algorithm through reinforcement learning by using a history of previous gradients and objective values as input for the agent. In [14], the authors improved the framework proposed in [13] for learning to optimize shallow neural networks. Bello et al. [3] proposed another approach for L2O by considering the problem of learning a controller that generates mathematical update equations from primitive functions, the controller was trained by RL. Faury and Vasile [7] proposed another

framework for L2O by splitting the optimizer into three sequential modules (update direction predictor, learning rate predictor, and resolution predictor); two were trained by RL in prototypical two-dimensional surfaces. Zhang et al. [24] proposed to learn a two-phase global optimization algorithm by deep learning and reinforcement learning. Silva-Filho [8] proposed to learn zeroth-order optimization algorithms by popular policy search algorithms.

3.3 Multi-Task Reinforcement Learning

Multi-task reinforcement learning is a subfield of reinforcement learning that deals with the problem of learning to solve multiple sequential-decision tasks at once [10,23]. It is related to L2O by Deep RL due to the characteristic of training a (Deep) RL agent in a distribution of similar tasks. Thus, we argue that L2O by Deep RL could benefit from the results of Multi-Task RL and vice-versa.

4 Learning Global Optimization by Deep RL

This section presents a framework for applying Deep RL to learn global optimization.

4.1 Problem Setup: L2O by Deep RL

In general, the problem setup is the same as the one discussed in 2.2. However, Deep RL requires a clear distinction between the agent and the environment. Furthermore, the environment can be formalized in different ways (e.g., POMDPs and MDPs).

Thus, we define the following problem setup for L2O by Deep RL:

- The agent is the optimizer $\mathcal{U}$;
- The environment is a distribution $\mathcal{P}_{\text{train}}$ of POMDPs;
- Each POMDP p in $\mathcal{P}_{\text{train}}$ represents the task of minimizing a function $f \sim \mathcal{F}_{\text{train}}$;
- For every $p_i, p_j \in \mathcal{P}_{\text{train}}$, the following properties hold:
  1. $\mathcal{A}_{p_i} = \mathcal{A}_{p_j}$, i. e., the action space is the same for all tasks;
  2. $\dim(o_t^{p_i}) = \dim(o_t^{p_j})$ for all $t \in \mathbb{N}$, i. e., all observations have the same dimensionality;
  - □ Note that the dynamics, emission function, rewards, state space, and observation space do not need to be equal.
- The agent's goal is to maximize $\mathbb{E}_{p \sim \mathcal{P}_{\text{train}}}\left[\bar{R}\right]$, that is, the agent should behave to maximize the average expected return across all training tasks.

This setup is closely related to multi-task RL [10] and meta-RL [23]; however, they differ mostly due to the nature of the tasks (i. e., MDPs instead of POMDPs).

How should each task $p \in \mathcal{P}_{\text{train}}$ be modelled? Up to this point, we have not discussed how to cast the task of minimizing an optimizee $f \sim \mathcal{F}_{\text{train}}$ as a POMDP. Most of the previous work in L2O by a Deep RL agent considered slightly different designs, for instance:

- [8] considered p as a MDP where
  - $a_t = \Delta\mathbf{x}$;
  - $s_t = o_t = \mathbf{x}_t$;
  - $R(s_t, a_t, s_{t+1}) = -f(\mathbf{x}_{t+1})$;
- [14] considered p as a POMDP where
  - $a_t = \Delta\mathbf{x}$;
  - $s_t = \{\mathbf{x}_t, \phi(\cdot)\}$, where $\phi(\cdot)$ are features that depend on previous $\mathbf{x}_i$, $\hat{f}(\mathbf{x}_i)$ (noisy optimizee values), and $\nabla\hat{f}(\mathbf{x}_i)$ (noisy gradients);
  - $o_t = \Psi(\cdot|s_t)$, where $\Psi(\cdot|s_t)$ are features based on the state s_t;
  - $R(s_t, a_t, s_{t+1}) = f(\mathbf{x}_t)$, and the agent's goal is to minimize the cumulative rewards;

We propose a new task design based on previous work in L2O for learning global optimization. Following the general problem of L2O, we consider the actions as the update in the optimizees variables (i. e., $a_t = \Delta\mathbf{x}$).

For the states s_t, we acknowledge that, due to the nature of global optimization problems, it is difficult to choose features that fully describe the state of the environment for all possible optimizees. For instance, if the problem is continuous, convex, and smooth, one can argue that knowing the gradients at a time-step t fully describes the information needed to choose an optimal action. However, if the problem is non-convex and multimodal, then solely knowing the gradient might not be enough to choose an appropriate action.

Thus, we propose to use $s_t = s_t^h \cup s_t^o$, where $s_t^h = \{\mathbf{x}_t, f(\mathbf{x}_t)\} \cup \mathbb{U}$ and $s_t^o = \{\nabla f(\mathbf{x}_i), f(\mathbf{x}_i), \mathbf{x}_i\}_{i=0}^{t}$. That is, the hidden state consists of the optimizee's value ($f(\mathbf{x}_t)$), variables ($\mathbf{x}_t$), and other features $\mathbb{U}$ that might or might not be available (e. g., known global optima, high-order information). On the other hand, the observable state consists of past values of the gradients, values, and variables.

This approach enables us to define different dynamics and reward functions, which use s_t, to better suit a given training optimizee, while keeping the same observation structure for all tasks.

We propose to consider the observations o_t as $\{\mathrm{P}(\nabla f(\mathbf{x}_t)), \boldsymbol{v}_t\}$, where $\mathrm{P}(\cdot)$ is the gradient processing function proposed in [1], and $\boldsymbol{v}_t$ is the agent's average velocity, inspired in [4]. The next subsection discuss how these features can be obtained from s_t^o.

Lastly, we propose the following general reward function:

$$R(s_t, a_t, s_{t+1}) = -\mathrm{T}(f(\mathbf{x}_{t+1})) \tag{2}$$

where $\mathrm{T} : \mathbb{R} \to \mathbb{R}$ is a transformation to scale $f(\mathbf{x}_{t+1})$. This reward function is similar to those in [13], [14], and [8]. We find it important for the rewards to be as close as possible to those of other tasks in $\mathcal{P}_{\text{train}}$. It is known in the multi-task RL setting that tasks with a greater expected return, in magnitude, are given more importance by the agent [10], thus individually scaling rewards is an important task. In the next section, we cover a possibility for T.

Since there are many choices of RL algorithms that either do not rely on the model or learn the model themselves [2], the dynamics are not described

here. The proposed task design aims to be as flexible as possible while capturing essential elements of global optimization, enabling future practitioners to adapt and improve it.

4.2 Incorporating Strategies for Improving Training

Given the problem setup defined previously, we propose to adapt and incorporate some of the training strategies discussed in 2.3 to the Deep RL setting.

Augmented training set. This strategy can be directly applied to the training optimizee distribution $\mathcal{F}_{\text{train}}$. We propose to use benchmark functions for global optimization as the base training set, $\mathbf{B}_{\text{train}} = \{f_1, \ldots, f_N\}$, and then define $\mathcal{F}_{\text{train}}$ as a distribution of these functions with some transformation parameters. That is, for $f, \phi \sim \mathcal{F}_{\text{train}}$, where $f \in \mathbf{B}$ and $\phi = \{\phi_0, \boldsymbol{\phi_1}, \phi_2\} \in \mathbb{R}$, the optimizee is $\phi_0 f(\mathbf{x} + \boldsymbol{\phi_1}) + \phi_2$ with $\phi_0, \phi_2 \in \mathbb{R}$ and $\boldsymbol{\phi_1} \in \mathbb{R}^n$.

Following this formulation, we can keep a small training set of benchmark functions with diverse characteristics, as proposed by [21], while being able to sample more optimizees, as proposed in [5].

Input features based on other optimizers. We propose to use two input features based on previous learned optimizers, and metaheuristics approaches.

The first input feature, $\mathrm{P}(\nabla f(\mathbf{x}_t))$, describes the gradient at the current time-step t, where P maps each $\nabla_k = \frac{\partial f}{\partial x_k} \in \nabla f$ to

$$\begin{cases} \left(\frac{\log(\nabla_k)}{10}, \operatorname{sgn}(\nabla_k)\right) & \text{if } |\nabla_k| \geq e^{-10} \\ \left(-1, e^{10}\nabla_k\right) & \text{otherwise} \end{cases}$$

which is the same gradient processing function proposed by [1].

The second input feature, inspired by [4] and metaheuristics approaches, describes the agent average velocity $\boldsymbol{v}_t$ up to the current time-step t. We define the average velocity as: $\boldsymbol{v}_0 = \mathbf{0}$, and $\boldsymbol{v}_t = \frac{\Delta \mathbf{x}}{\Delta t} = \frac{\mathbf{x}_t - \mathbf{x}_0}{t}$.

Normalization through scaling. We incorporate input normalization techniques directly into the input features, while $\mathrm{P}(\nabla f(\mathbf{x}_t))$ uses log and sgn for normalization, $\boldsymbol{v}_t$ is bounded by the action space $\mathcal{A}$, which is shared between all optimizees. However, it is also important to normalize returns and rewards besides normalizing inputs.

We propose to use the bi-symmetric log transformation [20] to scale the optimizees values. Thus, (2) can be rewritten as:

$$R(s_t, a_t, s_{t+1}) = -\operatorname{sgn}(y) \log_{10}(1 + |y \ln(10)|) \tag{3}$$

where $y = f(\mathbf{x}_{t+1})$, and ln is the natural logarithm. Perhaps, a more robust approach would be to adapt the RL algorithm to be scale invariant to the rewards magnitude, as proposed in [10], we leave this investigation to future work.

Curriculum Learning in a distribution of POMDPs. We propose to incorporate, inspired in [5], a curriculum strategy where the agent learns each task $p \in \mathcal{P}_{\text{train}}$ in an ordered manner. That is, the agent starts by learning simpler tasks (optimizees) and progressively learns more complex ones.

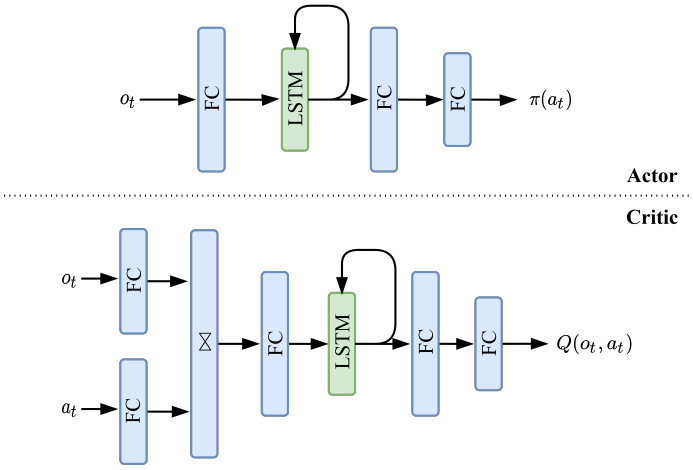

Fig. 1. Actor and Critic Networks. `FC` represents a fully connected layers, $\bowtie$ is a concatenation, and `LSTM` represents a long short-term memory layer. All layers have `relu` activation, except for the last fully connected layer of the actor (`tanh`), and critic (`linear`).

Each task $p \in \mathcal{P}_{\text{train}}$, is ordered by its optimizee's modality and separability. We refer to the pair {modality, separability} as a property c of an optimizee f_p. Modality is defined as the number of ambiguous peaks in the function surface [12], unimodal functions are easier to solve, while multimodal functions are harder. Separability is another characteristic that can be used to measure the difficulty of an optimizee [12]. Fully separable functions are easier to solve when compared to their non-separable counterpart.

Thus, the proposed curriculum partitions the total training (N episodes) in K sub-training stages, with n_i episodes each ($\sum_{i=1}^{K} n_i = N$), of progressive difficulty. For each sub-training stage i, the agent trains in tasks $p_0, \ldots, p_j \sim \mathcal{P}_{\text{train}}$ whose optimizees f_{p_k} have properties c_i. In this formulation, if $j > k$ for any $j, k \in [1, K]$, then an optimizee with properties c_j is harder to optimize than an optimizee with properties c_k.

In this work, we considered $K = 5$, and the following ordering: $c_1 = \{$unimodal, separable$\}$, $c_2 = \{$unimodal, partially-separable$\}$, $c_3 = \{$unimodal, non-separable$\}$, $c_4 = \{$multimodal, separable$\}$ and $c_5 = \{$multimodal, non-separable$\}$.

4.3 Recurrent TD3 for Learning to Global Optimize

To evaluate the proposed framework for learning global optimization, we selected the Twin Delayed DDPG (TD3) [9], a model-free actor-critic deep RL algorithm for solving MDPs, and a set of benchmark functions widely used in the optimization literature. We chose the algorithm TD3 due to its popularity, applicability in a variety of domains, and good preliminary results in previous L2O works [8].

Table 3. Composition of the Training set.

No.	Properties	Functions
3	Separable, Unimodal	Schumer Steiglitz, Sum Squares, Powell Sum.
8	Separable, Multimodal	Alpine 2, Csendes, Deb 1, Deb 3, Qing, Schwefel 2.26, W / Wavy, Weierstrass.
2	Partially-separable, Unimodal	Chung Reynolds, Schwefel.
6	Non-separable, Unimodal	Brown, Dixon Price, Schwefel 1.2, Schwefel 2.22, Schwefel 2.23, Streched V Sine Wave.
7	Non-separable, Multimodal	Exponential, Mishra 2, Salomon, Sargan, Trigonometric 2, Whitley, Zakharov.
Total: 26		

Table 4. List of evaluation functions.

Function	Properties	Search space	Global Minimum
Sphere	Unimodal	$x_i \in [0.0, 10.0]$	0.0
Bent Cigar	Unimodal	$x_i \in [0.0, 10.0]$	0.0
Ackley	Multimodal	$x_i \in [-35.0, 35.0]$	0.0
Griewank	Multimodal	$x_i \in [-100.0, 100.0]$	0.0
Rastrigin	Multimodal	$x_i \in [-5.12, 5.12]$	0.0
Rosenbrock	Multimodal	$x_i \in [-30.0, 30.0]$	0.0

In order to use TD3 to solve POMDPs, we needed to adapt its neural networks to learn to extract features from the past since the policies in POMDPs depend on past observations and actions (Subsection 2.2). In the RL literature, a simple approach is to add a Recurrent Neural Network (RNN) to the existing algorithms [11]. Figure 1 shows the architecture for the actor and critic networks.

5 Experiments and Analysis

This section discusses the experiments and analysis of the proposed approach for learning global optimization.

Training set. We consider a training set of 26 (Table 3) continuous, differentiable benchmark functions described in [12]. Each function is subject to its search space, and the optimizer must learn to optimize across different search spaces. The optimizee distribution $\mathcal{F}_{\text{train}}$ consists of those functions plus random transformations parameters ϕ.

Algorithm Hyperparameters. All hyperparameters have been empirically set by manual search. Unless stated otherwise, all results refer to the best-trained agent.

Evaluation. We first evaluate the learned optimizer in the training distribution, by sampling a 100 optimizees and comparing the average best value. Then, we evaluate the generalization of the optimizer in 6 benchmark functions (Table 4) not present in the training distribution. Unless stated other-

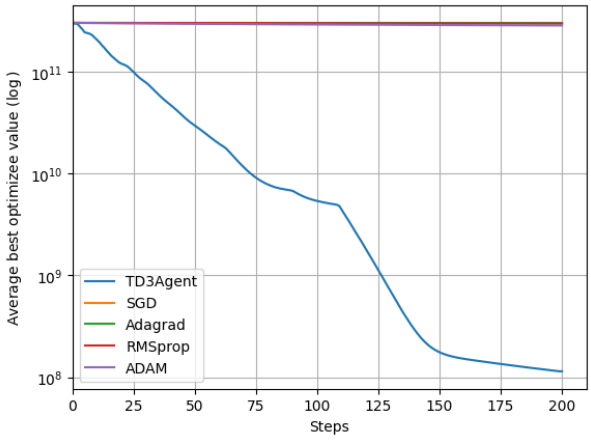

Fig. 2. Average best optimizee values over 200 steps for 100 sampled optimizees.

wise, both training and evaluation considered 2-dimensional optimizees (i. e., $d = 2$, and $\mathbf{x} = [x_0, x_1]^T$), with sampled transformation parameters in the range $\phi_0 \in [0.5, 1.5]$, $\boldsymbol{\phi_1} \in [-1.5, 1.5]^d$, $\phi_2 \in [-5.0, 5.0]$.

Baselines. To compare the proposed approach with other optimizers, we selected 4 first-order analytical optimizers, namely, Stochastic Gradient Descent (SGD), Adagrad, RMSprop, and ADAM. We used the default parameters set in TensorFlow for all problems.

Implementation details. We used Python 3.9, TensorFlow 2.7.0, and TF-Agent 0.11.0 in our implementations. Both training and evaluation were run in Google Colaboratory. The best learnt optimizer (agent) was trained for 2000 training episodes, with 75 steps each, with a training sequence length of 10 (i. e., size of sampled history from replay buffer). The curriculum strategy partitioned those 2000 episodes in: 200 episodes for c_1; 100 episodes for c_2; 100 episodes for c_3; 700 episodes for c_4; 900 episodes for c_5. During evaluation, the agent was given 200 steps to optimize the optimizees.

5.1 Comparison against Baselines

Figure 2 compares the average best optimizee values, in log scale, from 100 optimizees sampled from $\mathcal{F}_{\text{train}}$ over 200 steps. We found that none of the analytical optimizers were able to achieve results better than the learned optimizer `TD3Agent`.

Table 5 compares the best learned optimizer against the baselines, over a short horizon (200 steps), in the evaluation set. Since the evaluation functions were not available in the training set, we did not apply the transformations and evaluated the optimizers in the original function. We found that none of the algorithms were able to find the global optimum for the multimodal functions with precision; however, the `TD3Agent` found the best average solutions. It is

Table 5. Mean and standard deviation of best optimizee value over 100 independent runs for 2D problems with 200 steps.

Function	TD3Agent	SGD	Adagrad	RMSprop	ADAM
Sphere	**0.0** **(0.0)**	0.0232 (0.0144)	64.7831 (41.0723)	70.5634 (39.3927)	68.7703 (34.1719)
Bent Cigar	**0.0** **(0.0)**	2.9464866e7 (2.830116e7)	2.8709074e7 (2.7833004e7)	3.2879828e7 (3.0206872e7)	3.244124e7 (2.9399202e7)
Ackley	**16.9553** (5.4009)	19.0410 **(1.5761)**	20.3046 (2.4671)	19.4958 (2.2098)	19.3892 (2.3981)
Griewank	**1.5358** **(1.2512)**	1.9594 (1.0968)	2.5994 (1.2451)	2.4518 (1.2879)	2.3464 (1.1283)
Rastrigin	**6.1143** **(3.3523)**	14.0367 (5.4853)	37.4525 (15.0649)	22.0069 (12.5299)	22.7056 (12.9580)
Rosenbrock	**247.5643** **(538.5479)**	1.7007074e7 (2.1397274e7)	1.5635772e7 (2.0081116e7)	1.2053673e7 (1.850705e7)	1.1696214e7 (1.8787502e7)

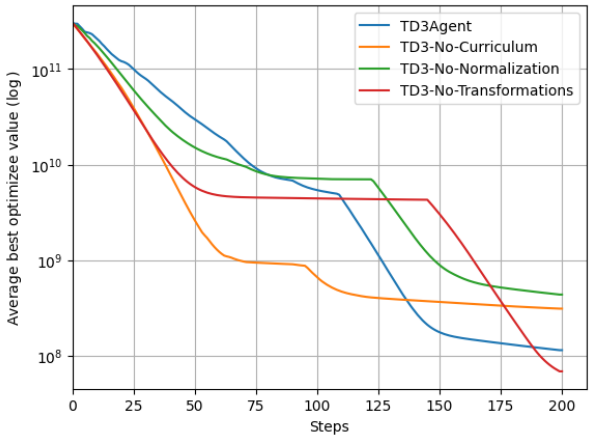

Fig. 3. Importance of different strategies to the learned optimizer.

also possible to see that the learned optimizer generalized well for the unimodal functions, but it could not fully generalize for the multimodal ones. We conjecture that no optimizee in the training set had a similar landscape to those in the evaluation set.

5.2 Ablation study of proposed techniques

To evaluate the importance of the proposed training strategies, we train 3 other agents, where one of the strategies is missing. Figure 3 shows the performance of the learned optimizers in 100 optimizees sampled from $\mathcal{F}_{\text{train}}$. We found that the most significant strategies were normalizing the rewards and applying a curriculum scheme.

6 Conclusion

Learning to optimize (L2O) is an auspicious field that has the potential to break many barriers in optimization and machine learning however, there are some issues to address first. For instance, learned optimizers are usually unstable to train, generalize poorly, and lack scalability. The L2O community has proposed different means to deal with those problems, such as novel training strategies and more sophisticated models. To further contribute with those strategies, we propose casting the L2O problem as a Deep Reinforcement Learning task, in which an agent must learn to solve a distribution of POMDPs, representing a distribution of minimization tasks. To this end, we construct a general framework for applying Deep RL to L2O and integrate training strategies found to improve other learnable optimizers. We evaluate the proposed framework to learn global optimization from training in benchmark functions and find that the learned optimizer was able to achieve better results than popular analytical optimizers. Through an ablation study, we find that normalizing rewards and applying a curriculum scheme were the most important strategies. We hope the contributions of this work lay a solid ground for future works with Deep RL in L2O.

7 Acknowledgements

The authors thank the Brazilian National Council for Scientific and Technological Development (CNPq) for the help during the development of this work.

References

1. Andrychowicz, M., Denil, M., Gómez, S., Hoffman, M.W., Pfau, D., Schaul, T., Shillingford, B., de Freitas, N.: Learning to learn by gradient descent by gradient descent. In: Advances in Neural Information Processing Systems. vol. 29 (2016)
2. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Process. Mag. **34**(6), 26–38 (2017)
3. Bello, I., Zoph, B., Vasudevan, V., Le, Q.V.: Neural optimizer search with reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 459–468. PMLR (2017)
4. Cao, Y., Chen, T., Wang, Z., Shen, Y.: Learning to optimize in swarms. In: Advances in Neural Information Processing Systems. vol. 32 (2019)
5. Chen, T., Zhang, W., Jingyang, Z., Chang, S., Liu, S., Amini, L., Wang, Z.: Training stronger baselines for learning to optimize. In: Advances in Neural Information Processing Systems. vol. 33, pp. 7332–7343 (2020)
6. Chen, Y., Hoffman, M.W., Colmenarejo, S.G., Denil, M., Lillicrap, T.P., Botvinick, M., de Freitas, N.: Learning to learn without gradient descent by gradient descent. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 748–756. PMLR (2017)
7. Faury, L., Vasile, F.: Rover descent: Learning to optimize by learning to navigate on prototypical loss surfaces. In: Learning and Intelligent Optimization. pp. 271–287. Springer International Publishing (2019)

8. Filho, M.S., Barbosa, G., Miranda, P., Nascimento, A., Mello, R.: Zeroth order policy search methods for global optimization problems: An experimental study. In: Anais do XVIII Encontro Nacional de Inteligência Artificial e Computacional. pp. 209–220. SBC (2021)
9. Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1587–1596. PMLR (2018)
10. Hessel, M., Soyer, H., Espeholt, L., Czarnecki, W., Schmitt, S., van Hasselt, H.: Multi-task deep reinforcement learning with popart. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 3796–3803 (2019)
11. Igl, M., Zintgraf, L., Le, T.A., Wood, F., Whiteson, S.: Deep variational reinforcement learning for POMDPs. In: Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 2117–2126. PMLR (2018)
12. Jamil, M., Yang, X.S.: A literature survey of benchmark functions for global optimisation problems. International Journal of Mathematical Modelling and Numerical Optimisation **4**(2), 150–194 (2013)
13. Li, K., Malik, J.: Learning to optimize. In: 5th International Conference on Learning Representations (2017)
14. Li, K., Malik, J.: Learning to optimize neural nets. CoRR (2017)
15. Lv, K., Jiang, S., Li, J.: Learning gradient descent: Better generalization and longer horizons. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 2247–2255. PMLR (2017)
16. Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., Sohl-Dickstein, J.: Understanding and correcting pathologies in the training of learned optimizers. In: Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 4556–4565. PMLR (2019)
17. Ni, T., Eysenbach, B., Salakhutdinov, R.: Recurrent model-free RL is a strong baseline for many pomdps. CoRR (2021)
18. Nobile, M.S., Cazzaniga, P., Ashlock, D.A.: Dilation functions in global optimization. In: 2019 IEEE Congress on Evolutionary Computation (CEC). pp. 2300–2307 (2019)
19. TV, V., Malhotra, P., Narwariya, J., Vig, L., Shroff, G.: Meta-learning for black-box optimization. In: Machine Learning and Knowledge Discovery in Databases. pp. 366–381. Springer International Publishing (2020)
20. Webber, J.B.W.: A bi-symmetric log transformation for wide-range data. Measurement Science and Technology **24**(2) (2012)
21. Wichrowska, O., Maheswaranathan, N., Hoffman, M.W., Colmenarejo, S.G., Denil, M., de Freitas, N., Sohl-Dickstein, J.: Learned optimizers that scale and generalize. In: Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 70, pp. 3751–3760. PMLR (2017)
22. Williams, R.J., Peng, J.: Function optimization using connectionist reinforcement learning algorithms. Connection Science **3**(3), 241–268 (1991)
23. Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., Levine, S.: Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In: Proceedings of the Conference on Robot Learning. Proceedings of Machine Learning Research, vol. 100, pp. 1094–1100. PMLR (2020)
24. Zhang, H., Sun, J., Xu, Z.: Learning to be global optimizer. CoRR (2020)