Parameters tuning and optimization for Reinforcement Learning algorithms using Evolutionary Computing

Franklin Cardeñoso Fernandez
Department of Electrical Engineering
Pontifícia Universidade Católica do Rio
Rio de Janeiro, Brazil
fracarfer5@aluno.puc-rio.br

Wouter Caarls
Department of Electrical Engineering
Pontifícia Universidade Católica do Rio
Rio de Janeiro, Brazil
wouter@ele.puc-rio.br

***Abstract*—Setting up the correct hyperparameters in reinforcement learning (RL) algorithms is an important part to achieve good performance in its execution and convergence. Manual adjustment for these hyperparameters is not a good practice because it consumes too much time and effort, therefore, it is advisable to use computational tools to optimize this tuning. Evolutionary computation (EC) techniques can be a good tool to tune and optimize the hyperparameters in the different algorithms. In this project we used the genetic algorithms (GA) approach to find the value of the hyperparameters that best fit the performance of the SARSA and Q-learning RL algorithms, addressing the underactuated pendulum swing-up task, maximizing the final rewards acquired and the agent's learning speed. We obtained good solutions with a fairly simple algorithm, but required multiple random restarts of the GA to escape local minima.**

***Index Terms*—genetic algorithms, reinforcement learning, hyperparameters, tuning.**

I. Introduction

Performing the selection of hyperparameters in different reinforcement learning RL algorithms is an important stage that affects their performance [1]. This hyperparameters adjustment can be done manually, verifying if the algorithm converged by testing the obtained policy once its execution has finished. However, performing manual adjustment is not a recommended practice because it consumes too much time and effort. For this reason the use of computational tools is advisable to perform hyperparameter tuning [2]; within these tools, a good option is to apply techniques of evolutionary computation EC for the tuning and optimization of the hyperparameters in the different algorithms, since these methods have shown good results in different tasks [3], [4], [5] and [6].

RL is a technique that bases its operation on the maximization of rewards received by an agent that interacts with its environment following a certain control policy[1]. To achieve the maximum reward it is necessary that the agent follow an optimal policy, which is evaluated and updated during the training of the agent as the result of its experience with the environment.

Within RL there are different methods and algorithms that are developed and used to solve different tasks. Within which, two of the algorithms most used are the SARSA and Q-learning algorithms for their easy implementation and understanding since they do not require a great computational effort for their execution and can be implemented easily in any programming language.

EC, contrary to RL bases its operation in evolution: where an initial population of individuals, under crossing and mutation processes produces descendants probably better, than the parents, and then performing the selection of best individuals. This process is performed iteratively a certain number of times until an optimal solution for the problem is found.

For this reason, in this project, we used genetic algorithms GA to perform the tuning of the hyperparameters of the algorithm SARSA(λ) and Q(λ), applied to the "underactuated pendulum swing- up" task, the algorithm operation is subsequently verified with the parameters found in a pendulum model simulator, selecting the parameters that maximize the end performance of the algorithm as well as stable operation. In this way, although it is not possible to perform this optimization in a real robot, we can perform it in a simulator and then learn in the real world with the found optimal parameters.

This paper is organized as follows: In section 2 a revision for previous work. Section 3 describes RL, EC and the pendulum problem. Section 4 and 5 present the methodology used and the results obtained respectively and finally, in section 6 and 7 we talk about the discussion, conclusions and future implementations.

II. Related work

Parameters tuning is not a recent problem, much research in this field have been done. For example, [7] perform parameter adjustment based on reliability applied in the grid world problem, or [8] which solve maze tasks applying Bayes inference to balance exploitation and exploration in model-based RL algorithms. In addition, [9] use Bayes estimation to apply particle filtering in order to estimate hyperparameters in RL model.

[10] presents a different approach using biological-based neuromodulators to tune hyperparameters in an adaptive form

in Markov decision tasks. Another interesting research is [11] which use GA for to find the step-size and the temperature decreasing hyperparameters of RL applied in a real robot.

In contrast, [12] explains about the influence on performance of the optimal hyperparameter selection for Deep Q-learning algorithms.

Finally, a more recent work [13] shows that is possible to combine the bayesian optimization with gaussian process to optimize hyperparameters of RL algorithms.

III. Theoretical Background

A. Reinforcement Learning

The main objective of RL is the reward maximization through the actions performed by an agent interacting with his environment in a certain state following a certain policy. During training, the agent changes his policy as the result of his experience with the environment [1]. Therefore, the objective of the agent is to maximize the expected rewards by optimizing a policy $\pi(s) \rightarrow a$ that maximizes the optimal action-value function:

$$Q^*(s,a) = E\{r_{t+1} + \gamma max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\}$$

Within RL, the algorithms most used for their simplicity are SARSA (λ) and $Q(\lambda)$ which learn on-policy and off-policy respectively. In the implementation of these algorithms appear different hyperparameters that determine their behavior, such as: the discount rate (γ), learning rate (α), decay rate (λ), exploration rate (ε), observations resolution and finally the initial values of the Q-table.

B. Evolutionary Computation

The area of evolutionary computation is that which incorporates in its algorithm the natural processes of mutation, crossing, reproduction and selection applied to the optimization of functions within a computer, thus achieving to use the power of the natural process of evolution as an alternative in the design of hardware and software[14]. In this way, this process is performed iteratively a certain number of times until an optimal solution for the problem is found by the evolution [15]. Considering that each individual of the population is a solution, the GA become a good alternative to search for solutions to optimization problems because they can address problems of non-linear, stochastic or discontinuous type, since it does not require a gradient generating a set of different solutions in each iteration in which the best solution approximates the optimal solution of the problem [16], [17].

C. The "Pendulum Problem"

One of the best known classical control problems is the "underactuated pendulum swing-up" task, which is a simple dynamic system used to evaluate and compare the performance of different algorithms of RL [18], the main one reason is because this system has only 2 dimensions, defined by the state: $x = [\varphi, \dot{\varphi}]$, where $\varphi = position$ and $\dot{\varphi} = velocity$, and a single control action: the voltage u of the pendulum motor.

The equation of motion of this system is given by:

$$\ddot{\varphi} = \frac{Mgl\sin(\varphi)}{J} - \frac{1}{J}(b + \frac{K^2}{R})\dot{\varphi} + \frac{K}{JR}u$$

Where the model parameters are: pendulum inertia (J), pendulum mass (M), gravity (g), pendulum length (l), damping (b), torque constant (K) and rotor resistance (R).

This system works in the following way: the pendulum starts in the resting state with the following state:

$$x_0 = [\pi, 0]$$

Where the position consists of the angle that the pendulum has with respect to the axis *Y* and the velocity consists of the angular velocity of the pendulum. In this way, the objective is to balance the pendulum in the state:

$$x = [0, 0]$$

Applying for this, a control signal (u) on the motor limited to $u \in [-3, 3]$ *V*, in order to keep the pendulum balanced. For this task, the following quadratic reward function is defined:

$$r(x_t, u_t) = -x_{t+1} Q x_{t+1}^\top - R u_t^2$$

with

$$Q = \begin{bmatrix} 5 & 0 \\ 0 & 0.1 \end{bmatrix}$$

and

$$R = 1$$

Which results in a quadratic function whose maximum value is at the point $x = [\varphi, \dot{\varphi}] = [\pi, 0]$ and which quadratically discounts the values of position, velocity and action different from 0.

IV. Experimental Methodology

One of the simplest ways to approach the pendulum task is to approximate the function-value through a process of discretization and use model-free dynamic programming (direct RL). For this project we used a MATLAB function which contains the Q-learning and SARSA algorithms implemented.

As mentioned before, the main objective of RL is to maximize the rewards obtained by the agent, however it is also necessary that the convergence method is stable, this means that the sum of rewards once the algorithm has converged is kept within a range without many variations. Taking these aspects in mind, in this case we have a multi-objective problem where we need to maximize the end performance and minimize the difference between the last 50 episodes. On the other hand, the GA Toolbox of MATLAB optimizes the functions using minimization of the objective function or also called *fitness function*.

We must therefore define several objectives in a single function, this was done by the following equation where under a single function is calculated the maximization of the end performance and the minimization of the error of the last episodes:

$$fitness\_function = -2(average\_reward) + error$$

Where *fitness_function* is the objective function, *average_reward* is the average reward obtained in the last 50 episodes and *error* is the Mean Square Error (MSE) of the reward in the last 50 episodes. As can be seen in the previous equations, another mathematical arrangement was made by multiplying by 2 the average reward to turn it into a goal with greater importance than the error.

Similarly, the constraints of the problem are defined. In this case, we are doing a tuning of hyperparameters, then, we need to define the numerical limits within which are the values of these parameters. The literature explains that the learning (α), decay (λ), exploration (ε) and discount (γ) rates are between 0 and 1 [1]. Therefore, the restrictions for these values are the following:

$$\begin{gathered} 0 < \alpha < 1, \\ 0 < \lambda < 1, \\ 0 < \varepsilon < 1, \\ 0 < \gamma < 1 \end{gathered}$$

Likewise, the other variables to be tuned are the resolution of discretization of the position, velocity and pendulum actions and the initial values of the Q-table. Because the position, speed and action are within the range: $[V_{min}, V_{max}]$, where $V_{min} = -\pi$ *rad* and $V_{max} = \pi$ *rad* for the position $V_{min} = -12\pi$ *rad/s* and $V_{max} = 12\pi$ *rad/s* for the speed and $V_{min} = -3$ *volts* and $V_{max} = 3$ *volts* for the action, it is possible to perform a discretization of these 3 variables so that each of them varies a value of Δstep each state, this is:

$$\Delta step = \frac{V_{max} - V_{min}}{Resolution}$$

and:

$$\begin{gathered} 1 \leq observations\_resolution \leq 100, \\ 1 \leq actions\_resolution \leq 10, \\ -1000 \leq Initial\_value \leq 1000 \end{gathered}$$

Where *observations_resolution* is the resolution for the variable of position and velocity, *actions_resolution* is the resolution of actions and *Initial_value* is the initial value of the Q table.

Finally, the Table I summarizes all the constraints and its type of data used for the construction of the algorithm. Note that we approximated the open limits by closed limits with offset $\Delta = 10^{-5}$.

TABLE I
CONSTRAINS FOR GA OPTIMIZATION

Variable	Minimum value	Maximum value	data type
α	$0 + \Delta$	$1 - \Delta$	*float*
λ	$0 + \Delta$	$1 - \Delta$	*float*
ε	$0 + \Delta$	$1 - \Delta$	*float*
γ	$0 + \Delta$	$1 - \Delta$	*float*
obs_res	1	100	*integer*
action_res	1	100	*integer*
initial_value	-1000	1000	*integer*

Finally, with the restrictions, the limits and the objective function defined, were implemented the fitness function, the evaluation function and the main evolution program. The main program creates the initial population within the constraints and send the initial parameters to the fitness function where is calculated the objective value in function of the average reward added with the MSE of the last 50 episodes gotten from the RL program until the stopping criteria is reached. Next, the founded values are tested with the evaluation function for 10 times in order to get the mean algorithm behaviour and get average reward, rise time and MSE values. So, this action is continually repeated until the error is as small as be possible. Finally, once the program is finished, the algorithm behaviour with the best parameters is ploted in a graph to show the performance obtained.

V. RESULTS

After having carried out different experiments, the implemented algorithm was able to find many set of hyperparameters for this task. Tables II and III show the five sets of hyperparameters with highest fitness found by the optimization procedure.

TABLE II
BEST HYPERPARAMETERS OBTAINED FOR SARSA ALGORITHM

Exp	α	γ	ε	λ	obs	act	$init\_Q$
1	0.1171	0.9383	0.0140	0.8453	34	3	54
2	0.1959	0.9571	0.0347	0.7943	32	3	-216
3	0.0685	0.9147	0.0262	0.9192	34	3	18
4	0.3054	0.9712	0.0274	0.7069	28	3	-320
5	0.9992	0.9172	10^{-5}	0.5572	40	3	0

TABLE III
BEST HYPERPARAMETERS OBTAINED FOR Q-LEARNING ALGORITHM

Exp	α	γ	ε	λ	obs	act	$init\_Q$
1	0.6711	0.9891	0.0004	0.4212	50	3	-86
2	0.8944	0.9377	0.0056	0.2879	75	3	23
3	0.4738	0.9406	0.0136	0.5115	75	3	40
4	0.4594	0.9479	0.0257	0.4940	74	3	-28
5	0.5811	0.9784	0.0434	0.3986	75	5	-65

Their respective performances are shown in Tables IV and V, and are quite close together when compared per algorithm, but with sometimes very different hyperparameters. In addition we can confirm this with the learning curve graphs showed in Figures 1 and 2. In particular, the learning rate α and trace decay rate λ are not very sensitive. The same holds for the initialization, although it should have a value around 0. Interestingly, we observe that the optimal number of actions is quite low, even though theoretically we could achieve best performance increasing the actions. We attribute this to the limited number of episodes used for training. We are performing 1000 episodes, therefore, with more actions, we are not be able to guarantee that the algorithm manage to learn a reasonable policy during training.

In addition, we notice that the optimal observations resolution found are quite different for both algorithms. In the

SARSA algorithm this value is around 28-40 contrary to Q-learning which optimal value is around 50-75. Perhaps SARSA, which calculates the value function of the noisy exploration policy, cannot take advantage of a more accurate value function representation.

TABLE IV
PERFORMANCE OBTAINED FOR SARSA ALGORITHM WITH THE HYPERPARAMETERS FOUND

Experiment	*End Performance*	*MSE*	*Rise Time*
1	−621.99	42.294	551
2	−641.87	16.485	206
3	−659.83	66.117	634
4	−674.67	0.0063	142
5	−688.14	0	175

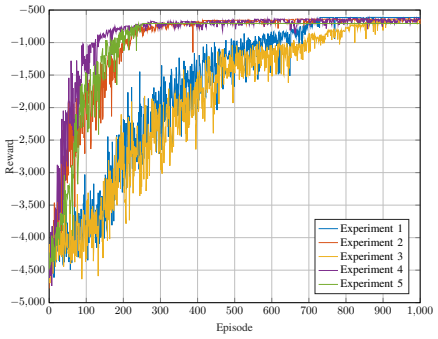


Fig. 1. Performance obtained for SARSA algorithm

TABLE V
PERFORMANCE OBTAINED FOR Q-LEARNING ALGORITHM WITH THE HYPERPARAMETERS FOUND

Experiment	*End Performance*	*MSE*	*Rise Time*
1	−553.37	0	357
2	−570.10	0	571
3	−570.1	0	875
4	−571.54	2.054	664
5	−574.28	1.893	709

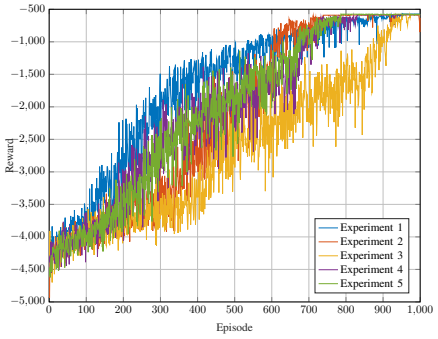


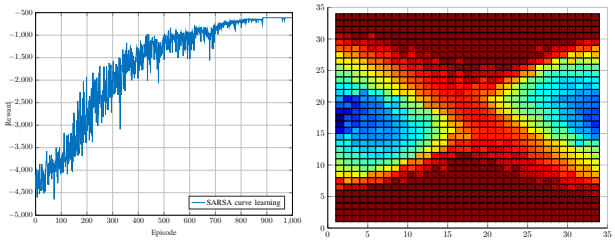
Fig. 2. Performance obtained for Q-learning algorithm

Finally, Table VI show the hyperparameters for the best results found, in addition, Fig. 3 and 4 show convergence

TABLE VI
BEST RESULTS FOR SARSA AND Q-LEARNING

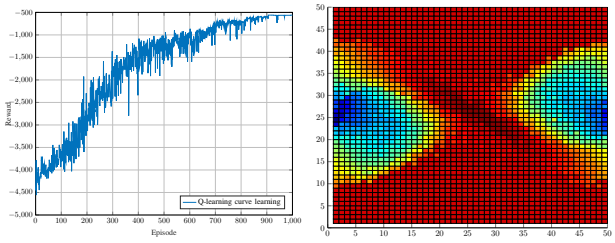
Variable	SARSA	Q-learning
α	0.1171	0.6711
γ	0.9383	0.9891
ε	0.0140	0.0004
λ	0.8453	0.4212
obs_res	34	50
action_res	3	3
initial_value	54	−86

and value function of SARSA and Q-learning for the best set chosen and showing a curve learning comparison in Fig 5 respectively.



(a) Learning curve (b) Value function

Fig. 3. Best results for SARSA algorithm



(a) Learning curve (b) Value function

Fig. 4. Best results for Q-learning algorithm

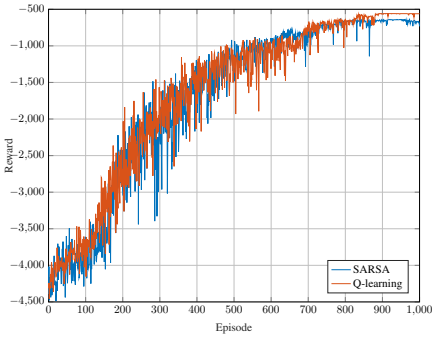


Fig. 5. Performance comparison for SARSA and Q-learning algorithm

VI. DISCUSSION

The different results from Section V were each the outcome of a full GA run, and present the five best runs from a total of

850 such runs for each experiment (SARSA and Q-learning). It is clear that the best fitness achieved from a single GA run is quite noisy, indicating many local minima. In fact, many runs got stuck in minima that did not manage to swing up the pendulum at all. As such, we recommend the use of GAs with automatic restart strategies [19].

Looking further at the parameters of the best 40 runs of the Q-learning experiment, we can see in Figure 6 that the learning rate α and trace decay rate λ are negatively correlated: a higher α implies a lower λ. This is expected because both increase the update to the value function. The correlation between α and the exploration rate ε is also expected. α acts as a noise filter with lower values causing smaller updates and therefore more filtering. Higher ε increases the exploration noise, and therefore requires more filtering.

A less well-known correlation exists between the discount rate γ and the value function initialization, shown in Figure 6. A higher γ implies a lower initialization. Initializing the value functions with high values encourages exploration [1]. Since all rewards are negative, a higher γ reduces the overall value function, reducing the required initialization in order to achieve the same amount of exploration.

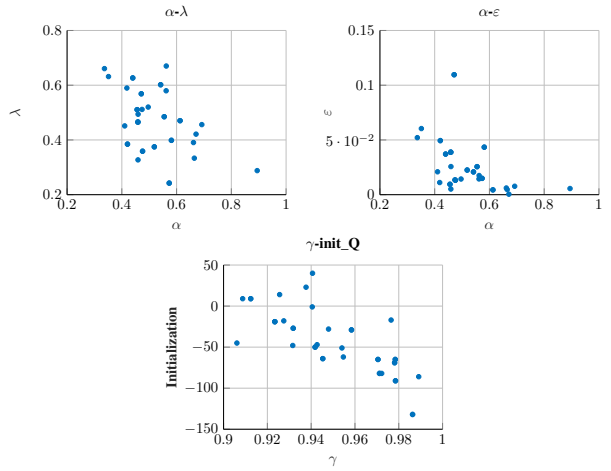


Fig. 6. Correlation for hyperparameters obtained for Q-learning algorithm

VII. Conclusion and Future Work

As presented in this paper, the use of EC techniques to find the different hyperparameters of the RL algorithms is a recommended alternative capable of obtaining good results with a fairly simple algorithm. On the other hand, it was necessary to implement a multi-restart function in order to have diversity in the results and avoid falling into local minima during the process of optimization and search of parameters.

Finally, is necessary to improve the multi-restart function and the GA algorithm and test with another RL algorithms in order be able to make a better comparison of results as we said before. This implementation is pending for future implementations.

Acknowledgment

Thanks to Prof. Marco Aurelio Cavalcanti Pacheco for the guidance in Evolutionary Computation for the realization of this project.

References

[1] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proceedings of the 24th International Conference on Neural Information Processing Systems*, ser. NIPS'11. USA: Curran Associates Inc., 2011, pp. 2546–2554. [Online]. Available: http://dl.acm.org/citation.cfm?id=2986459.2986743

[3] L. Padierna, M. Carpio, A. Rojas Dominguez, H. Soberanes, R. Baltazar, and H. Fraire-Huacuja, "Hyper-parameter tuning for support vector machines by estimation of distribution algorithms," pp. 787–800, 12 2017.

[4] S. R. Young, D. C. Rose, T. P. Karnowski, S.-H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, ser. MLHPC '15. New York, NY, USA: ACM, 2015, pp. 4:1–4:5. [Online]. Available: http://doi.acm.org/10.1145/2834892.2834896

[5] B. Yuan and M. Gallagher, "A hybrid approach to parameter tuning in genetic algorithms," in *2005 IEEE Congress on Evolutionary Computation*, vol. 2, Sept 2005, pp. 1096–1103 Vol. 2.

[6] D. E. Moriarty, A. C. Schultz, and J. J. Grefenstette, "Evolutionary algorithms for reinforcement learning," *J. Artif. Int. Res.*, vol. 11, no. 1, pp. 241–276, Jul. 1999. [Online]. Available: http://dl.acm.org/citation.cfm?id=3013545.3013551

[7] H. Hayama and Y. Mutoh, "Adjustment of meta-parameter in reinforcement learning based on reliability," *Proceedings of the Japan Joint Automatic Control Conference*, vol. 51, pp. 253–253, 2008.

[8] S. Ishii, W. Yoshida, and J. Yoshimoto, "Control of exploitation–exploration meta-parameter in reinforcement learning," *Neural Networks*, vol. 15, no. 4, pp. 665 – 687, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608002000564

[9] K. Samejima, K. Doya, Y. Ueda, and M. Kimura, "Estimating internal variables and parameters of a learning agent by a particle filter," in *Advances in Neural Information Processing Systems*. MIT Press, 2004, p. 04.

[10] N. Schweighofer and K. Doya, "Meta-learning in reinforcement learning," *Neural Netw.*, vol. 16, no. 1, pp. 5–9, Jan. 2003. [Online]. Available: http://dx.doi.org/10.1016/S0893-6080(02)00228-9

[11] A. Eriksson, G. Capi, and K. Doya, "Evolution of meta-parameters in reinforcement learning algorithm," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 1, Oct 2003, pp. 412–417 vol.1.

[12] N. Sprague, "Parameter selection for the deep q-learning algorithm," 2015.

[13] J. C. Barsce, J. A. Palombarini, and E. C. Martínez, "Towards autonomous reinforcement learning: Automatic setting of hyper-parameters using bayesian optimization," *CoRR*, vol. abs/1805.04748, 2018. [Online]. Available: http://arxiv.org/abs/1805.04748

[14] J. Foster, "Evolutionary computation," *Nature Reviews Genetics*, vol. 2, pp. 428 – 436, 2001, evolutionary Computation. [Online]. Available: https://www.nature.com/articles/35076523.pdf

[15] A. Eiben and M. Schoenauer, "Evolutionary computing," *Information Processing Letters*, vol. 82, no. 1, pp. 1 – 6, 2002, evolutionary Computation. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0020019002002041

[16] Matworks. (2018) Algoritmo genetico. [Online]. Available: https://la.mathworks.com/discovery/genetic-algorithm.html

[17] C. H. North, J. A. Joines, M. G. Kay, C. R. Houck, and C. R. Houck, "A genetic algorithm for function optimization: A matlab implementation," Tech. Rep., 1996. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.4413

[18] I. Grondman, "Online model learning algorithms for actor-critic control," 01 2015.

[19] A. S. Fukunaga, "Restart scheduling for genetic algorithms," in *International Conference on Parallel Problem Solving from Nature*. Springer, 1998, pp. 357–366.