Improving the performance of Continuous Action Reinforcement Learning Automata

Abdel Rodríguez[1,2], Matteo Gagliolo[2], Peter Vrancx[2], Ricardo Grau[1], and Ann Nowé[2]

[1] Bioinformatics Lab, Universidad Central de Las Villas, Santa Clara, Cuba
[2] AI Lab, CoMo, Vrije Universiteit Brussel, Brussels, Belgium
abdelr@uclv.edu.cu

Abstract. Learning automata (LA) are policy iteration reinforcement learners that exhibit nice convergence properties in discrete action settings. Recently, a novel formulation for continuous action reinforcement learning automata was proposed (CARLA), featuring an interesting exploration strategy, which is well suited for learning from sparse reward signals. In this paper, we propose an improvement of the algorithm, and evaluate its impact on performance comparing with the original version, as well as with another continuous LA. The experimental evaluation is carried out both in simulation and on a real control setup, showing a clear advantage of our method, which also performs well in distributed control settings.

1 Introduction

Learning automata are reinforcement learners, belonging to the category of policy iterators. Their built-in exploration strategy makes them very well suited for distributed and multi-agent settings. In a discrete action setting, it has been proven that a team of independent learners using a LA with reward-inaction update converges to a Nash Equilibrium [5]. This applies to both common interest as well as conflicting interest n-player, m-action games. Here, we use independent learners, meaning that the agents select actions independently and only observe their own feedback, and not the actions taken by other agents, nor their feedback.

Despite the interesting convergence properties for these discrete action LAs, many applications require the ability to deal with continuous actions. This has lead to the introduction of continuous action LA (CALA) [4] and continuous action reinforcement learning automata (CARLA) [2]. In these continuous schemes, the policy is represented as a probability distribution over actions, which is updated after each epoch. In CARLA, the policy is represented as a non-parametric distribution, initially uniform over action space. After each epoch, the probability of the executed action is reinforced, mixing the current density function with a Gaussian bell centered in the action itself, whose amplitude is proportional to the observed reward. More specifically, CARLA is a stateless learner capable of optimizing over a compact set given the possibly nose corrupted observations of certain function.

A mayor drawback of the technique, however, is its computational complexity. Recently, Rodríguez et al. [3], analyzed the CARLA approach, and proposed a significant

improvement in terms of computation time and convergence to the global optimum. The improved automaton performs better on easy-to-learn functions, still it does not perform very well in noisy or even noiseless scenarios with very small variance in the resulting feedback for the learner.

In this paper, we present a further improvement to the CARLA approach, analyzing its positive impact on performance in both single and multi-dimensional settings. We introduce new ideas for making the learner more sensitive to small changes near the optimum, but still absorbing noise in order to converge to globally optimal actions.

The formulation of CARLA used in this paper is described in Section 2. Section 3 introduces our modifications to the CARLA scheme. Section 4 reports experimental results, and Section 5 concludes the paper.

2 Learning Automata

The learning automaton is a simple model for adaptive decision making in unknown random environments. Engineering research on LA started in the early 1960's [6]. Tsetlin and his colleagues formulated the objective of learning as an optimization of some mathematical performance index [5]:

$$J(\Lambda) = \int_A R(\mathbf{a}, \Lambda)\, \mathrm{d}P(\mathbf{a}) \tag{1}$$

where $R(\mathbf{a}, \Lambda)$ is the (possibly stochastic) reward function, Λ is a parametric representation of the distribution over actions $P(\mathbf{a})$, $\mathbf{a}$ is the action, and A is the set of possible actions, assumed to be compact. The performance index J is the expectation of R with respect to the distribution P, and it therefore includes randomness in $\mathbf{a}$ and randomness in R. The problem of maximizing J is analogous to the well-known *multi-armed bandit* problem, with discrete or continuous arm set [1].

In this section we recall two alternative versions of continuous action LA. In both cases, the current policy of the automata is represented as a probability distribution over the action set, from which actions are drawn sequentially. The learning process consists of updating this distribution, based on the observed reward, with the aim of increasing the probability of actions which achieve the highest rewards.

The first algorithm, which we will use as a comparison in our experiments, is the Continuous Action Learning Automata (CALA) algorithm, introduced in [4]. The authors implement P as a normal probability distribution with mean μ_t and standard deviation σ_t. At every time step t, an action is selected according to a normal distribution $N(\mu_t, \sigma_t)$. Then, after exploring action a_t and observing reward signal $\beta_t(a_t)$, the update rules (2) and (3) are applied, resulting in new values for μ_{t+1} and σ_{t+1}.

$$\mu_{t+1} = \mu_t + \lambda \frac{\beta_t(a_t) - \beta_t(\mu_t)}{\max(\sigma_t, \sigma_L)} \frac{a_t - \mu_t}{\max(\sigma_t, \sigma_L)} \tag{2}$$

$$\begin{aligned} \sigma_{t+1} = \sigma_t &+ \lambda \frac{\beta_t(a_t) - \beta_t(\mu_t)}{\max(\sigma_t, \sigma_L)} \left[\left(\frac{a_t - \mu_t}{\max(\sigma_t, \sigma_L)} \right)^2 - 1 \right] \\ &- \lambda K(\sigma_t - \sigma_L) \end{aligned} \tag{3}$$

where λ is the learning parameter controlling the step size ($0 < \lambda < 1$), K is a large positive constant, and σ_L is a lower bound of the standard deviation σ_t. The authors also derived a convergence proof for this automaton, and tested it in games among multiple learners.

Rodríguez et al. [3] remarked the poor applicability of this method in practical problems. Note that the update rule needs two evaluations of the reward function β_t, one at the selected action a_t, one at the mean of the probability distribution μ_t. Sampling both actions is not practical, especially in scenarios where the evaluation of the reward function is costly, or takes a relatively long time, as it is often the case with electro-mechanical systems. The convergence can also be hindered by the noisiness of the reward function β_t.

We now describe an alternative LA implementation, the Continuous Action Reinforcement Learning Automaton (CARLA) [2], on which our method is based. In CARLA, P is implemented as a nonparametric distribution, initially uniform over the action space A, assumed to be a compact set. After exploring action $a_t \in A$ in time step t, the probability density function (pdf) f_t is updated as in (4), where parameter λ_t is the spreading rate, determining the size of the neighborhood of a_t which will be reinforced, and α is a learning rate. Notice that sampling the mean μ_t is no longer necessary.

$$f_{t+1}(a) = \begin{cases} \gamma_t \left(f_t(a) + \beta_t(a_t)\, \alpha e^{-\frac{1}{2}\left(\frac{a-a_t}{\lambda_t}\right)^2} \right) & a \in A \\ 0 & a \notin A \end{cases} \tag{4}$$

At each iteration t, an action is drawn from the current pdf. To achieve this goal, the cumulative density function (CDF) is required. Since the pdf used in CARLA is nonparametric, it is expensive to re-evaluate the CDF every time the function changes. In order to significantly reduce the computational complexity, [3] introduced an alternative update rule which operates directly on the CDF. This version of CARLA, which will be used in our experiments, is described in (5), where α is again the learning rate, λ_0 is the initial spreading rate, β is the reward function, i.e., the objective that the learner has to maximize, a_t is the action selected in time-step t, σ_t is the standard deviation of the actions selected recently, and $F_{N(0,1)}$ is the standardized normal cumulative density function.

$$F_{t+1}(a) = \begin{cases} 0 & a < A_{min} \\ \gamma_t \left(F_t(a) + \delta_t D_t(A_{min}, a_t) \right) & a \in A \\ 1 & a > A_{max} \end{cases} \tag{5}$$

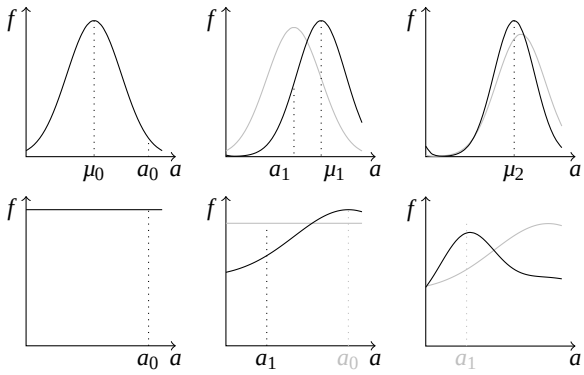
where:

$$\begin{aligned} \delta_t &= \beta_t(a_t)\, \alpha \lambda_t \sqrt{2\pi} \\ \gamma_t &= \frac{1}{1 + \delta_t D_t(A_{min}, A_{max})} \\ D_t(x,y) &= F_{N(0,1)}\left(\frac{y-a_t}{\lambda_t}\right) - F_{N(0,1)}\left(\frac{x-a_t}{\lambda_t}\right) \\ \lambda_t &= \lambda_0 \frac{12\sigma_t}{(A_{max} - A_{min})^2} \end{aligned} \tag{6}$$

Both CALA and CARLA are policy iterators: in practice, the main difference among these two algorithms resides in the way the action space is explored. In CALA, which

employ a Gaussian distribution over actions, the exploration is locally distributed around the mean of the current distribution; in CARLA, exploration is based instead on an initially uniform, nonparametric distribution. The exploration can cover different areas of the action space, and, depending on the reward function, it can converge to an arbitrary, possibly multimodal, distribution. The CARLA update scheme exhibits interesting exploration characteristics in environments with sparse reward signals. This is on the one hand due to the action probabilities being initially uniform, and on the other hand to the fact that the sampling of action that leads to zero reward results in a reduction of the selection probabilities in the neighborhood of such action. This forces the exploration of actions outside the penalized area. If the next selected action again leads to no reward, the area around that action will again be penalized, and the exploration driven towards areas that have not been sampled yet. When dealing with sparse reward signals, policy iteration methods typically get "stuck", as there is no gradient information to guide further search. As soon as a CARLA learner picks up a reward, instead, the probability density around the corresponding action is increased, and convergence towards the optimum can start, analogous to policy gradient methods. In conclusion, CARLA methods allow exploratory sampling in the beginning of the search process in a similar way as value iteration approaches, with a reward scheme where each unsuccessful action, i.e. not making a direct transition into the goal state, results in negative immediate reward and guides the exploration towards different regions. Fig. 1 shows illustrative examples of the exploration strategies of the CALA and CARLA algorithms.

Fig. 1: Sketches of LA action space exploration, for CALA (above) and CARLA (below).



3 CARLA Online Reward Transformation

In this section we introduce our modification of the CARLA algorithm. The basic idea is to improve the algorithm's convergence, by rescaling the reward signal to amplify differences in the feedbacks. From the analysis introduced by Rodríguez et al. [3], it can

be seen that larger differences in the feedbacks for different actions, lead to faster convergence to the optimum. If the reward function varies very little around the optimum, many time-steps may be required to detect the difference in feedback, and converge to the optimum. If a learner memorizes the last few rewards, it can analyze their range, in order to linearly transform newly received rewards, viewing them as a high or low signals relative to recently observed rewards. Since exploration is still possible, *z-score standardization* is very useful to deal with this rescaling, avoiding sensitivity to outlier observations. Let μ_{rwd} and σ_{rwd} be the mean and standard deviation of the rewards collected, then every reward can be transformed as shown in expression (7). We stress that μ_{rwd} and σ_{rwd} should be calculated from untransformed rewards.

$$\beta'_t(a_t) = \max\left(0, 0.5 + \frac{\beta_t(a_t) - \mu_{rwd}}{2\sigma_{rwd}}\right) \tag{7}$$

Note that negative values are not allowed, in order to prevent the probability density function from becoming negative. Using this transformation, it is easier for the learner to differentiate rewards, no matter how close to the optimum it is exploring. Also observe that this is not a way to introduce extra information, it just modifies the reward function increasing the amplitude of the difference between the extreme values of this function. Scaling this amplitude does not imply changing the location of the maximum (or minimum) values.

Fig. 2: Transformation of rewards. Both charts plots the reward functions. X axis represents the action and Y axis the value of the function. We can see the original, untransformed reward function on the left. The black curve shows the reward function. On the right we plot both the original and transformed reward function in gray and black, respectively.

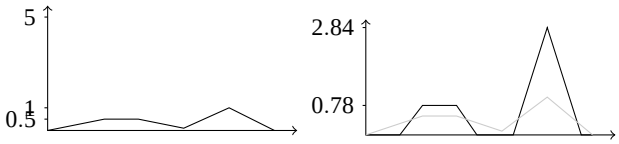


Figure 2 shows an illustrative example of our method. A learner exploring around the local maximum on the left will have difficulty detecting the slightly better action to the right. The figure on the right shows the effect of our transformation. The black curve represents β' while the gray curve corresponds to the original function. Note that the only change is in the amplitude of the function, the maxima are still located at the same points. Since the exploration was concentrated on the local maximum to the left, this is the zone that has been "zoomed in". Nonetheless, the other maximum situated to the right has also been amplified. Any exploratory actions to the right will now clearly detect the better feedback. Of course, in a real learning setting we could not simply transform the entire reward function offline, and we would instead need to calculate the transformation online, based on the current sample mean and standard deviation.

4 Experimental Results

In this section, we validate the two proposed modifications for CARLA with several experiments, also comparing with the performance of CALA. We first consider several single-automaton function optimization scenarios, both noiseless (Sec. 4.1) and noisy (Sec. 4.2). We then demonstrate the performance of our algorithm on a simple yet challenging control problem. Finally, we show that the algorithm also performs well in multi-automaton settings. Parameter settings are fixed as follows: for CALA, λ, σ_L and K were set to 0.01, 0.03 and 5 respectively, while the initial distribution is the standard normal with $\mu_0 = 0$ and $\sigma_0 = 1$. For CARLA, $\alpha = 0.1$ was used as a learning rate, and $\lambda_0 = 0.2$ as initial spreading rate.

4.1 Noiseless Settings

We recall the three example target functions β^i,β^{ii} and β^{iii} (8), introduced by [3]. These three reward functions are displayed in Figure 3.

$$\begin{aligned} \beta^i(a_t) &= \mathrm{b}(a_t, 0.5, 0.2) \\ \beta^{ii}(a_t) &= 0.9\,\mathrm{b}(a_t, 0.2, 1) \\ \beta^{iii}(a_t) &= (0.9\,\mathrm{b}(a_t, 0.2, 0.4)) \bigcup \mathrm{b}(a_t, 0.9, 0.3) \end{aligned} \tag{8}$$

with

$$x \bigcup y = x + y - xy$$
$$\mathrm{b}(a, a_0, \sigma) = e^{-\frac{1}{2}\left(\frac{a-a_0}{\sigma}\right)^2}$$

Fig. 3: Reward functions β^i (very simple function), β^{ii} (slowly varying with the action) and β^{iii} (with two optima)

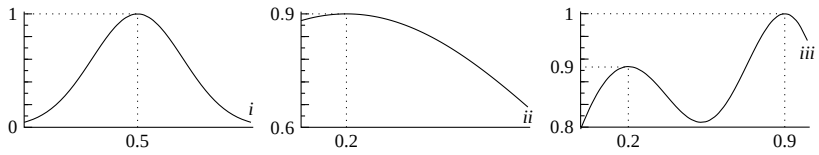
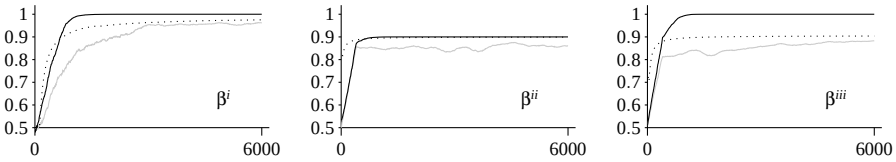


Figure 4 shows the performance obtained by the CARLA with the online reward transformation (ORT) algorithm (black curve) and the CARLA algorithm as described in [3], without transformation of rewards (gray curve). The dotted line represents the performance of the CALA algorithm. Remember that the latter requires two function evaluations for each update (see Section 2). In this and following plots, the horizontal axis reports time in terms of the number of reward function evaluations, while the vertical axis shows the obtained average reward over time. The plotted rewards are the untransformed ones. It is clear that the transformation of rewards modification displays the best performance. The chart in the middle, representing function β^{ii}, shows the lowest difference, with the dotted (CALA) line barely below the black (CARLA-ORT) line. For the β^i and β^{iii} functions, the CARLA-ORT algorithm shows a marked improvement over both standard CARLA and CALA.

Fig. 4: Average rewards for β^{i}, β^{ii} and β^{iii}, versus number of reward function evaluations. The black and gray lines report the performances of CARLA, with and without ORT, respectively, while the dotted line corresponds to CALA.



4.2 Noisy Settings

In the following experiment, we add noise to the reward functions introduced in the previous section. The noisy functions are shown in (9), where $\text{rand}(r) \in [0,r]$ is a uniformly distributed random number. We also consider two additional reward functions (10), where two functions with different optima are mixed with equal probability. Figure 5 plots these two functions, the dotted line representing the expected reward.

$$\begin{aligned} \beta^{i'}(a_t) &= 0.8\,\beta^{i}(a_t) + \text{rand}(0.2) \\ \beta^{ii'}(a_t) &= 0.7778\,\beta^{ii}(a_t) + \text{rand}(0.2) \\ \beta^{iii'}(a_t) &= 0.8\,\beta^{iii}(a_t) + \text{rand}(0.2) \end{aligned} \tag{9}$$

$$\begin{aligned} \beta^{iv}(a_t) &= \begin{cases} 0.8\,\text{b}(a_t, 0.2, 0.3) & \text{rand}(1) < 0.5 \\ \text{b}(a_t, 0.9, 0.2) & \textit{otherwise} \end{cases} \\ \beta^{v}(a_t) &= \begin{cases} \text{b}(a_t, 0.8, 0.2) & \text{rand}(1) < 0.5 \\ 0.8\,\text{b}(a_t, 0.5, 0.3) & \textit{otherwise} \end{cases} \end{aligned} \tag{10}$$

Fig. 5: Reward functions β^{iv} and β^{v}. The functions are composed by two bell shaped functions that are randomly selected to generate the reward. The dotted line represents the expected reward.

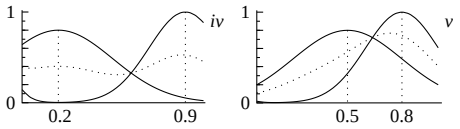


Figure 6 shows the average rewards collected over time. As was the case before, the black curve shows the results obtained using transformation of rewards, the gray curve displays the performance of CARLA without ORT, and the dotted line corresponds to CALA. The results obtained here are similar to those reported in the previous subsection, with the CARLA-ORT algorithm outperforming the other approaches.

Fig. 6: Average rewards for $\beta^{i'}$, $\beta^{ii'}$, $\beta^{iii'}$, β^{iv} and β^{v}. The gray curve represents the standard CARLA while the black one the results obtained with the new proposal. The dotted line corresponds to CALA.

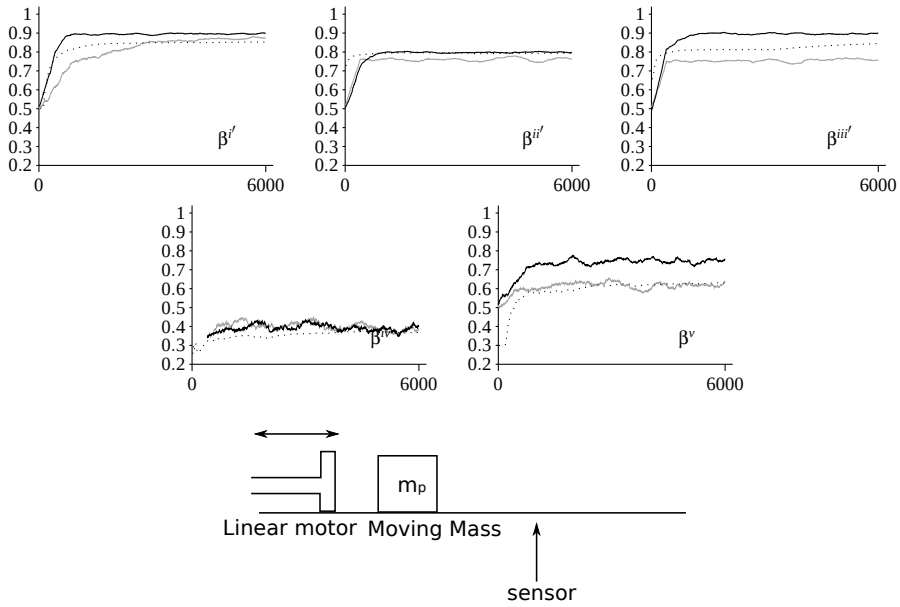


Fig. 7: Sketch set-up, position measurement at one location

4.3 Control Task

In many industrial applications, it is either not possible or too expensive to install sensors which measure the system's output over the complete stroke: instead, the motion can only be detected at certain discrete positions. The control objective in these systems is often not to track a complete trajectory accurately, but rather to achieve a given state at the sensor locations (e.g. to pass by the sensor at a given time, or with a given speed). Model-based control strategies are not suited for the control of these systems, due to the lack of sensor data. In this section, we present experiments with a simple abstraction of such systems, an electro-mechanical setup consisting of a linear motor and a moving mass mounted on a straight horizontal guide (Figure 7). The position of the moving mass is monitored via a single discrete sensor, set along the guide, which fires at the passage of a small (1 cm) element attached to the mass. When the motor is activated, the mass is "punched" forward, and slides up to a certain position, depending on the duration and speed of the motor's stroke, and on the unknown friction among the mass and its guide. Two tasks are defined on this setup, with different objectives: a) let the mass pass the sensor at a predefined time (time task); b) let the mass stop exactly in front of the sensor (position task).

In the first series of experiments, we only consider constant speed signals, with duration varying on a closed interval, such that an action consists of a single scalar,

which we normalize in $[0,1]$, an epoch consists of a single action, followed by a scalar reward, and no state information is used. For each task, a corresponding reward function is implemented, favoring the desired behavior. For the time task, given a target time t_0, reward is given as $r = \exp\{-c(t-t_0)^2\}$, where c is a constant ($c = 1000$), and t is the time at which the sensor starts firing, which is ∞ if the mass does not reach it. For the position task, the reward is given as the portion of time during which the sensor fires, over a fixed time interval (4*s*), measured from the beginning of the motor's movement. Fig. 8 reports samples of the two reward functions: note that the system is highly stochastic, and repeating the same action may result in different rewards.

The results of the modified CARLA algorithm are shown in Figure 9 on both tasks. We plot single experiments with the actual setup, each lasting for 300 epochs. CARLA was run with a learning rate of 0.8, and a constant spreading rate 0.25. The lower bar plot reports the returns observed at each epoch, while the upper plot reports the actions performed (+), along with the evolution of statistics describing the policy: mean and standard deviation of the CARLA nonparametric distribution. The position task turns out to be more difficult: this can easily be explained comparing the reward samples (Fig. 8). The best action for the position task, around 0.4, is a "needle in a haystack" compared to the time task, where the reward function changes more gradually around the optimal action. Nonetheless, the CARLA algorithm is able to converge to a good action for both tasks.

4.4 Multi-automata systems

In this section we evaluate the performance of our algorithm on a number of multi-automata settings, where several LA learn in the same environment. At every time step, each automaton selects a single scalar, corresponding to one of the dimensions of the action vector. The reward obtained by the automata therefore depends on all input actions.

In a first series of experiments, we apply CARLA and CARLA-ORT to a number of characteristic functions with 2 input actions. These functions were chosen to study the behavior of the algorithm in settings where coordination is required. In the first three experiments, the automata have to coordinate on a narrow region in the 2D action space, using only a noisy reward signal. The fourth experiment investigates the result of individual reward functions for the two automata. The automata still have to coordinate to receive a reward, but have different preferences for the selected combined action. Each automaton wants to minimize the magnitude of its own action, but when both do this at the same time the result is a 0 reward. This is intended to model a control task with multiple controllers, where each controller tries to minimize an individual cost (e.g. energy use). The results for these functions are shown in Figure 10.

$$
\begin{aligned}
\beta^{i}(a_t,b_t) &= 0.8\,|a_t+b_t-1| + \text{rand}(0.2)\\
\beta^{ii}(a_t,b_t) &= 0.8\,|a_t+b_t-1| - \tfrac{b_t}{10} + \text{rand}(0.2)\\
\beta^{iii}(a_t,b_t) &= \begin{cases} 1 & |a_t+b_t+N(0,0.1)-1| < 0.2 \\ 0 & otherwise \end{cases}\\
\beta^{iv}_{a}(a_t,b_t) &= \begin{cases} 1-\frac{a_t}{10} & |a_t+b_t+N(0,0.1)-1| < 0.2 \\ 0 & otherwise \end{cases}\\
\beta^{iv}_{b}(a_t,b_t) &= \begin{cases} 1-\frac{b_t}{10} & |a_t+b_t+N(0,0.1)-1| < 0.2 \\ 0 & otherwise \end{cases}
\end{aligned}
\tag{11}
$$

To conclude, we consider a 2D input version of the control tasks of the previous section, in which one automaton controls the speed of the motor, and the other controls the duration of the signal. The reward functions for both automata are identical to those described above, but the outcome of the system (either the time the sensor fires or final position of the mass, depending on the task) now depends on the values selected by both automata. Figure 11 shows how, on both tasks, the automata succeed in coordinating on a pair of actions which leads to high rewards.

5 Conclusions

We introduced CARLA-ORT, an improvement of continuous action reinforcement learning. The new algorithm was evaluated on a number of noiseless and noisy function optimization scenarios, as well as on a real control setup, outperforming both the original CARLA, and an alternative continuous action learner (CALA). It also performed very well in distributed control settings, where each agent had to learn a different dimension of the control signal. Ongoing research is aimed at improving the coordination among different agents in the distributed setting, in order to address more challenging open loop control applications.

References

1. S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-Armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.
2. M. N. Howell, G. P. Frost, T. J. Gordon, and Q. H. Wu. Continuous action reinforcement learning applied to vehicle suspension control. *Mechatronics*, 7(3):263 – 276, 1997.
3. A. Rodríguez, R. Grau, and A. Nowé. Continuous action reinforcement learning automata. performance and convergence. In Joaquim Filipe and Ana Fred, editors, *3rd International Conference on Agents and Artificial Intelligence ICAART 2011*, pages 473 – 478. SciTePress, January 2011.
4. G. Santharam, PS Sastry, and MAL Thathachar. Continuous action set learning automata for stochastic optimization. *Journal of the Franklin Institute*, 331(5):607–628, 1994.
5. M. A. L. Thathachar and P. S. Sastry. *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Kluwer Academic Publishers, 2004.
6. M. Tsetlin. The behavior of finite automata in random media. *Avtomatika i Telemekhanika*, pages 1210–1219, 1962.

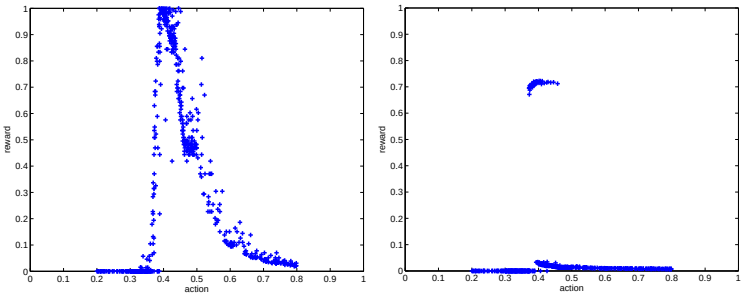


Fig. 8: Reward functions for the two control tasks, sampled for 500 randomly chosen actions on a subset of the unit interval, including the optimal action. Left: time task. Right: position task.

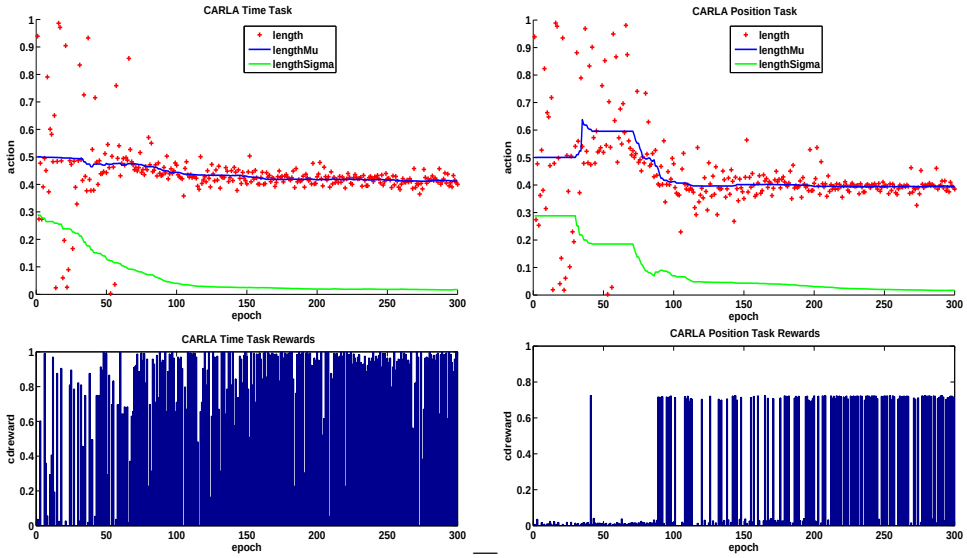
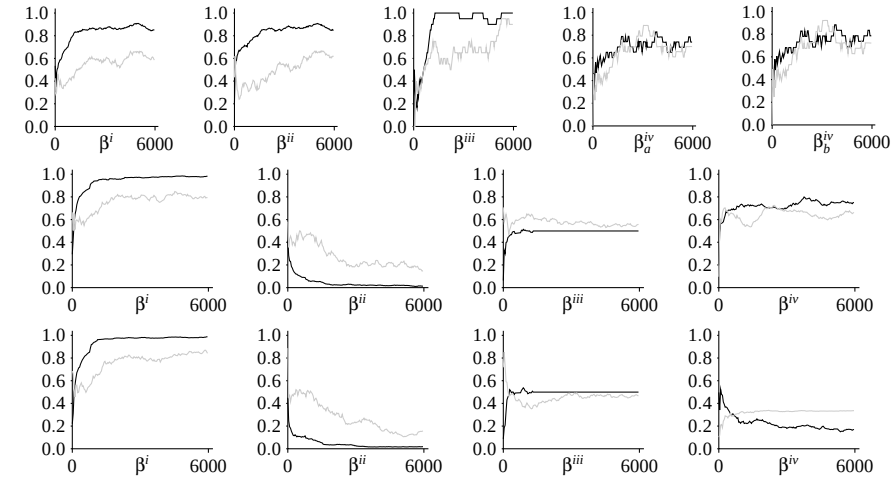


Fig. 9: Results for the CARLA-ORT algorithm on both control tasks. The top row shows sampled actions together with together with the mean and standard deviation of the action probabilities. Bottom row shows obtained rewards. Left: time task. Right: position task.

Fig. 10: Results for the multi-automata reward functions in Equation 11. The gray curve represents the standard CARLA while the black one the results obtained with the new proposal. First row charts show the average rewards collected. Second and third rows show the mean of the sampled actions for players one and two respectively. All averages are calculated using a sliding window of 20 samples.



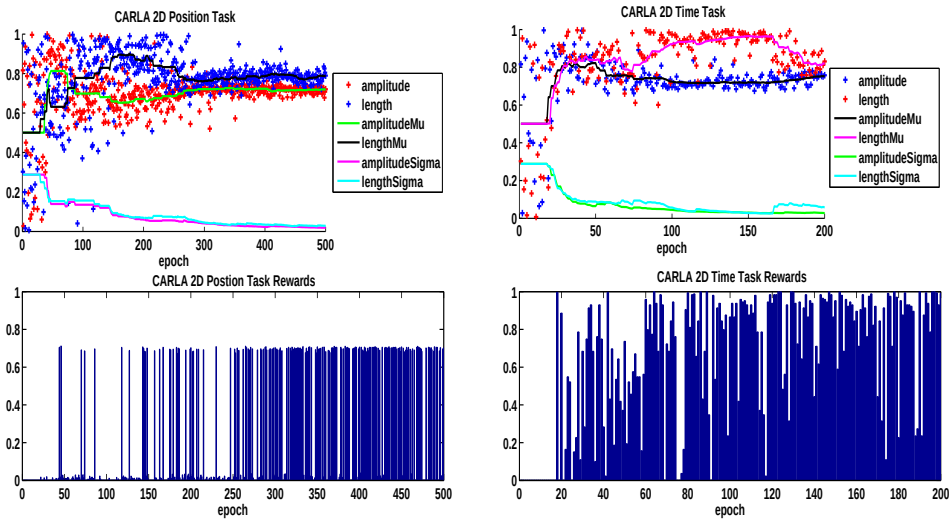


Fig. 11: Results for the CARLA-ORT algorithm on both 2D control tasks. The top row shows sampled actions together with the mean and standard deviation of the action probabilities. Bottom row shows obtained rewards. Left: time task. Right: position task.