Reinforcement learning using continuous states and interactive feedback

Angel Ayala
angel.ayala@alumnos.ucentral.cl
Universidad Central de Chile
Santiago, Chile

Claudio Henríquez
chenriquezb@ucentral.cl
Universidad Central de Chile
Santiago, Chile

Francisco Cruz
francisco.cruz@ucentral.cl
Universidad Central de Chile
Santiago, Chile

ABSTRACT

Research in intelligent systems field has led to different learning methods for machines to acquire knowledge, among them, reinforcement learning (RL). Given the problem of the time required to learn how to develop a problem, using RL this work tackles the interactive reinforcement learning (IRL) approach as a way of solution for the training of agents. Furthermore, this work also addresses the problem of continuous representations along with the interactive approach. In this regards, we have performed experiments with simulated environments using different representations in the state vector in order to show the efficiency of this approach under a certain probability of interaction. The obtained results in the simulated environments show a faster learning convergence when using continuous states and interactive feedback in comparison to discrete and autonomous reinforcement learning respectively.

KEYWORDS

Reinforcement learning, continuous interactive reinforcement learning, Q-learning

ACM Reference Format:
Angel Ayala, Claudio Henríquez, and Francisco Cruz. 2019. Reinforcement learning using continuous states and interactive feedback. In *Proceedings of ACM International Conference Proceedings Series (APPIS2019)*. ACM, New York, NY, USA, Article 37, 5 pages.

1 INTRODUCTION

Nowadays, the technology is giving us the ability to perform repetitive tasks of great effort by robotics mechanisms programmed to execute a specific activity, for example, in a linear production factory there are a variety of robotic structures, located one after the other, that complement their functions in order to produce a specific product. However, there are more complex activities that online production systems are not able to perform under a classical methodology of algorithm development [5], and it is necessary to implement artificial intelligence techniques, which provide the necessary knowledge to the system for the execution of tasks [6].

Reinforcement learning (RL) algorithms are a computational approach to learning from interaction, which is focused on learning through goal-oriented interaction [8]. These algorithms provide the ability for robots (or automated systems) to perform tasks of greater complexity and they should, as the humans, learn to execute them in order to achieve the goal. However, these algorithms have a performance problem, because excessive time is used in order to achieve an acceptable learning and, therefore, the implementation is also expensive [5].

In this work, we present an algorithm as an extension of RL which allows the interaction with another agent using a continuous vector state called interactive reinforcement learning (IRL). We have implemented a simulated scenario in order to better test the proposed algorithms. Although the IRL approach speeds up the learning process for classic RL, there is additional motivation to explore other features in order to improve the convergence using IRL with a continuous state representation.

2 REINFORCEMENT LEARNING AND INTERACTIVE FEEDBACK

Reinforcement learning (RL) [8] refers in the one hand to animal psychology, defining learning as the result of trial and error, and on the other hand, solves the problem of optimal control through value functions and dynamic programming. However, it also involves methods of temporal differentiation, which are the beginning of the modern field of RL.

An RL method is an effective way to solve Markov Decision Processes (MDP) problems, which have a close relationship with the problems of optimal control, where the dynamic programming allows to approach the learning by means of the successive iteration of approximations to the correct answer. The RL definition is based on the principle of Effect Law [10], which establishes: from many responses made in front of the same situation, those which entail a satisfaction for the animal, will allow a strengthening between the action and the situation, and increases the probability that it repeats that action for that situation. On the contrary, if its response entails discomfort, this link between action and situation is weakened, decreasing the probability that it will happen again.

Based on the animal learning context, the term reinforcement appears after Thorndike's Principle of Effect Law, which establishes the strengthening of behavior patterns as the result of an animal receiving a stimulus in a temporal relationship with another stimulus or response, where some psychologists enlarge the concept with the weakening of these patterns, which generate a change in the animal behavior [8].

The idea of trial and error, under a computational approach, appeared for the first time in a 1948 report, in which Alan Turing describes a pleasure-pain system design that works under the principle of the Effect Law: "When a configuration is reached for which the action is undetermined, a random choice for the missing data is made and the appropriate entry is made in the description,

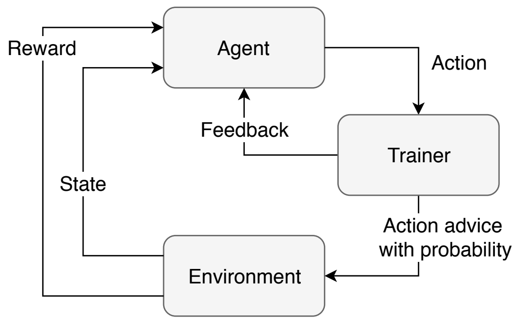

Figure 1: Interactive reinforcement learning with policy shaping diagram.

tentatively, and is applied. When a pain stimulus occurs, all tentative entries are canceled, and when a stimulus occurs, they are all permanent." [11].

2.1 Markov decision processes

In Markov Decision Processes (MDP), the state vector obtained from the environment must provide all the necessary information [8], such as sensor measurements, but it may contain much more than this. This status representation can be a complex structure composed of a sequence of sensations. This is called Markov Property.

An agent-environment interface can be implemented with this representation of the state, where the actor or agent interacts with its environment [8] to perform a task. This interaction is made through a transition function that, for a certain action executed by the agent, modifies the observable environment and gives a reward signal. Therefore in order to solve an MDP through RL, it must define a space of state, actions, a transition function, and a reward function.

2.2 Interactive feedback

This learning method is an extension of reinforcement learning, which includes an external trainer that offers instructions in order to optimize decision-making [4], these instructions will act as a guide for the learner through the feedback strategy [9], which can be through the configuration in the policy or the modification in the reward signal.

The policy shaping approach, used in this work, can be seen in Figure 1. In this approach, the action proposed by the apprentice can be replaced by a better action, chosen by the external trainer, before the execution [3]. This action replacement may occur given a certain feedback probability $\mathcal{L}$ which determines the frequency of advice delivered by the trainer.

2.3 Continuous reinforcement learning

In conventional RL, most of the time, are only considered MDP with discrete states and actions space [12], however, in many real-world applications the discretization of this action space is not very useful since generalization becomes more difficult from past experiences and learning becomes slow. A way to learn from a continuous domain is Q-learning[14], which allows the agent learning to act in an optimal manner by experiencing actions and consequences. It is an incremental method of dynamic programming which consecutive updates improve the chosen actions in specific states. Moreover, another approach is an algorithm capable of managing a continuous space of states and action is the Continuous Actor-Critic Automaton (CACLA) that has the ability to find continuous real solutions, a better generalization of properties and a quick selection of action as shown in [15].

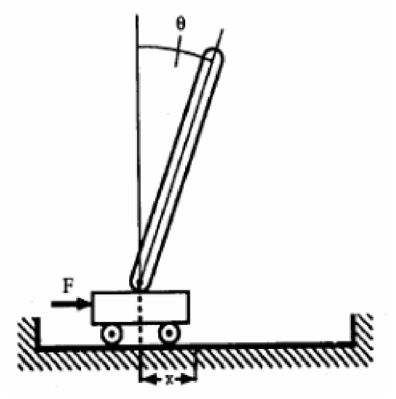

Figure 2: Pole balancing problem [2]

3 EXPERIMENTAL SETUP

The scenario is based on the Pole Balancing Problem (see Figure 2), which requires a closed-loop feedback controller, whose behavior must balance a pole connected to a motor-driven car. The movement of the car is restricted to a movement of a horizontal axis by means of a track, and the post is free to move on this axis through a pivot [2].

3.1 Pole balancing MDP

For the implementation of the RL algorithm, the next elements accordingly to the MDP are defined :

- **States**: The state vector has a continuous representation conformed by $< x, x', \theta, \theta' >$ corresponding, respectively, to the position of the car with respect to the center of the track, the speed of the car, the angle of inclination of the post with the car, and the angular velocity of the post.
- **Actions**: The actions have a discrete representation corresponding to the direction in which the car should move on the track, the *left* that executes the force in that direction and *right* that executes a force in the opposite one.
- **Transition Function**: To obtain a variation of the state vector, a force of constant magnitude obtained from an equation that has the four variables of the state vectors as inputs to the system must be applied to the car; this variation is obtained by using differential equations typical of the physical model of the environment as shown in [2].
- **Reward Function**: As long as the agent holds the pole in a vertical position, a reward equal to 1 is awarded, and if it drops, or goes beyond the boundaries of the track, the reward is equal to 0.

4 DESIGN AND IMPLEMENTATION OF THE AGENTS

For the aforementioned environment, the implementation of the agents is done in simulated setup, which should be able to obtain a high reward while the episodes progress. In the case of the Pole Balancing Problem, two agents are implemented, one of them is able to interact through a discrete state vector, and the other one using a continuous states vector.

It is necessary for the agent to design the way in which it decides to take actions in his environment where an optimal balance between exploitation and exploration of the space of actions is required, therefore the information available for this decision depends on the actions taken previously. Thus the agent must explore the space of action compensating the good actions already explored with others that it has never tried before [6]. For the solution of this compensation between exploitation and exploration, there has been implemented the ϵ-greedy method which has an exploration rate ϵ, randomly chosen in a normal distribution [5], and this strategy generally results in a greater reward without getting caught in a local minimum [8].

For the first approach to this algorithm, the off-policy Q-learning method [13] has been implemented, since this method of learning allows the agent the ability to act optimally, through the experience of the consequence of the actions without the need to build a map in the Markovian domain.

4.1 Interactive Approach

A first IRL approach is made through the implementation of an external agent, the trainer, which gives feedback with an action, according to its policy configuration, to a second agent given a probability of feedback $\mathcal{L}$. Therefore, the algorithm extends the RL agent, modifying the way in which the action is selected involving the trainer's policy as shown in Algorithm 1. This implementation is done for the agents of both environments in their respective representations.

Algorithm 1 IRL Agent

```
function SELECTACTION($s_t$)
    if randomValue < $\mathcal{L}$ then
        $a_t \leftarrow$ from trainer's Q function for $s_t$
    else
        $a_t \leftarrow$ from agent's policy for $s_t$
    return $a_t$
```

4.2 Discrete Representation

A first approach to the implementation of the algorithm for the pole balancing tackles its discrete representation, in order to simplify the complexity in finding an optimal learning policy. Since the state vector of the system has continuous values, it must be discretized in ranges with upper and lower limits. The BOXES approach [7] divides the state space of the system into discrete regions called partitions or boxes to reduce the problem complexity afterward, the method updates the action for each partition (box).

In the implementation of Q-learning [13] for this type of representation, a Q matrix is generated that stores the reward obtained for each pair of action-state, allowing the simplification in the estimation of the value function, validating of "how good" is the action taken given a certain state, the definition "good" is based on terms of reward or return expected for a given policy [8] this can be observed in Algorithm 2.

Algorithm 2 Discrete Agent

```
1:  Initialize Q matrix with uniform weights[-1, 1]
2:  for Episode = 1, $M$ do
3:      Observe $s_0$ from enviroment
4:      Discretize $s_0$ with BOXES
5:      while $s_t$ not terminal do
6:          Select action $a_t$ using $\epsilon$-greedy policy
7:          Observe $r_t, s_{t+1}$
8:          Discretize $s_{t+1}$ with BOXES
9:          Update Q matriz with tuple $s_t, a_t$
10:         $s_t \leftarrow s_{t+1}$
11:     Reduce exploration rate ($\epsilon$) and learning rate ($\alpha$)
```

4.3 Continuous representation

Since the continuous representation belongs to a whole range of real numbers, the memory cost in order to generate a Q matrix for all possible states is unfeasible, therefore, for the solution in continuous spaces is used an approximator function [12]. This function allows estimating the Q-value from the state vector without the need to store it as a value within the matrix but as a Q-function.

To implement the algorithm in this representation, two multilayer perceptron is used; one is used for the training of the agent and the second to compute the expected value for the Q-function. In addition, a Temporal-Difference learning optimization is implemented where a tuple is stored in a memory of $<$ state$_t$, action$_t$, reward$_t$, state$_{t+1}$ $>$ that is used to obtain the setting for the training of the neural networks [8].

Each neural network has a 4x24x24x2 architecture, which has 4 inputs corresponding to the state vector, 2 hidden layers with 24 neurons each, and 2 outputs representing the actions that the agent can select. The initial weights are defined randomly from a normal distribution and the training control corresponds to the mean square error, with a linear output function. This method is known as deep Q-network [14] implemented in Algorithm 3.

5 TRAINING ANALYSIS AND COMPARISON

In order to obtain a correct comparison and results, a total of 50 agents were trained, with their respective configurations for the convergence of learning as follows.

For the simulated environment, the CartPole-v1 was used from the OpenAI Gym library [1], which defines the maximum duration for each episode in 500 units of time, is the maximum value that can be obtained as a reward; a minimum reward value of 475 indicates that the task was successfully executed for the episode during the agent training. When the successful execution is completed in 50 consecutive episodes, it is considered that the task was learned.

Algorithm 3 Continuous Agent

```
1: Initialize memory $D$ with large $N$
2: Initialize Q function with uniform weights[-1, 1]
3: Initialize Q approximator with uniform weights[-1, 1]
4: for episode = 1, M do
5:     Observe $s_0$ from the enviroment
6:     while $s_t$ not terminal do
7:         Select action $a_t$ using $\epsilon$-greedy policy
8:         Observe $r_t, s_{t+1}$
9:         Store the tuple $(s_t, a_t, r_t, s_{t+1}$, terminal) in D
10:        Update Q function from $D$
11:        $s_t \leftarrow s_{t+1}$
12:    Copy weights from Q function to Q approximator
13:    Reduce value of exploration rate ($\epsilon$).
```

Table 1: Discrete representation agent's parameters.

Parameter	Value
Discount Factor (γ)	0.99
Exploration Rate (ϵ)	[1, 0.01]
Learning Rate (α)	[0.5, 0.1]

Table 2: Continuous representation agent's parameters.

Parámetro	Valor
Discount Factor (γ)	0.99
Exploration Rate (ϵ)	[1, 0.005]
Learning Rate (α)	0.001

5.1 Discrete representation results

Table 1 the parameters used during the learning. The parameters, except for the discount factor, have a decreasing value between episodes from an initial value to a fixed minimum value. The variation of the parameters is defined by equation (1).

$$MAX(param_{max}, MIN(param_{min}, param - \log_{10}(\frac{\text{ep} + 1}{25}))) \quad (1)$$

The results obtained with the previous configuration (Figure 3) show where the convergence of learning from episode 150 on, allowing the agent to improve its performance in the following episodes, being able to get the maximum reward since episode 247, from where it can be maintained vertical for more than 50 consecutive episodes. For this representation, the agent's interactive approach shows that the convergence of learning is reached from episode 100 onwards, however, the agent manages to execute the task satisfactorily since episode 232, from where it is able to maintain the post vertical for more than 50 consecutive episodes.

5.2 Continuous representation results

For the algorithm with continuous representation only the exploration rate parameter decrease over time, as in equation (2), and shown in table 2.

$$param_{t+1} = param_t * 0.999 \quad (2)$$

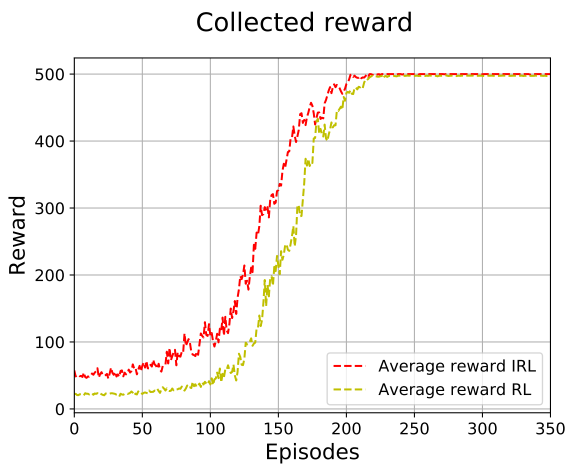

Figure 3: Results of training of 50 agents for the environment CartPole-v1 discretization state vector with BOXES[7], and feedback probability of 0.3.

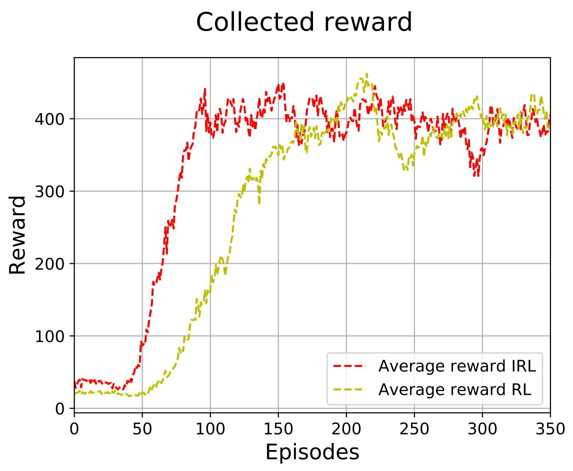

Figure 4: Results of training of 50 agents for environment CartPole-v1 with continuous representation and feedback probability of 0.3.

As shown in Figure 4, for this representation the autonomous RL agent converges after 55 episodes and reaches a maximum reward of 460 in episode 215, being unable to remain stable for the following episodes. However, under the interactive approach, there is a faster convergence of learning which starts from episode 40 and reaches a maximum value of reward of 450 in episode 155.

The previous results indicate that although the interactive approach does not exceed the reward values obtained during the training, this approach allows a faster convergence of learning. Moreover, the use of an advisor also benefits the convergence, nevertheless, it is crucial to have good advice in order to take advantage of it, otherwise, inconsistent advice may be detrimental for the learning process.

6 CONCLUSIONS

In this work, we have shown that an interactive agent with a continuous state space is able to obtain a faster convergence of learning and perform a fewer number of episodes compared to the autonomous agent for the same problem. In this interactive approach, by using a continuous state space , the agent achieves a faster learning convergence.

Although the discrete representation achieves a more stable performance in terms of reward this may not be implemented in all real-world scenarios.

Moreover, the interactive approach for the reinforcement learning agent allows a better convergence respect to the autonomous approach using the same probability of policy feedback.

For future work, the implementation of an IRL agent in continuous action space is considered. For instance, using the algorithm Continuous Actor-Critic Learning Automation [12], it is expected an improvement in terms of the time needed for the convergence of learning.

7 ACKNOWLEDGEMENTS

The authors gratefully acknowledge partial support by Universidad Central de Chile under the research project CIP2017030.

REFERENCES

[1] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. 2016. OpenAI Gym. *ArXiv e-prints* (June 2016). arXiv:1606.01540

[2] J. Brownlee. 2005. *The pole balancing problem: A benchmark control theory problem.* Technical Report 7-01. Swinburne University of Technology, Melbourne, Victoria, Australia.

[3] T. Cederborg, I. Grover, C. L. Isbell, and A. L. Thomaz. 2015. Policy Shaping with Human Teachers. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, 3366–3372. http://dl.acm.org/citation.cfm?id=2832581.2832718

[4] F. Cruz, S. Magg, Y. Nagai, and S. Wermter. 2018. Improving interactive reinforcement learning: What makes a good teacher? *Connection Science* 30, 3 (2018), 306–325. https://doi.org/10.1080/09540091.2018.1443318 arXiv:https://doi.org/10.1080/09540091.2018.1443318

[5] F. Cruz, S. Magg, C. Weber, and S. Wermter. 2014. Improving reinforcement learning with interactive feedback and affordances. In *4th International Conference on Development and Learning and on Epigenetic Robotics*. 165–170. https://doi.org/10.1109/DEVLRN.2014.6982975

[6] S. Marsland. 2009. *Machine Learning: An Algorithmic Perspective.* (1st ed.). Chapman and Hall/CRC, New York.

[7] D. Miche and R. A. Chambers. 1968. 9 BOXES: An experiment in adaptive control.

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

[9] A. L. Thomaz and C. Breazeal. 2006. Reinforcement Learning with Human Teachers: Evidence of Feedback and Guidance with Implications for Learning Performance. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1 (AAAI'06)*. AAAI Press, 1000–1005. http://dl.acm.org/citation.cfm?id=1597538.1597696

[10] E. L. Thorndike. 1911.. *Animal intelligence; experimental studies,*. New York,The Macmillan company,. 324 pages. https://www.biodiversitylibrary.org/item/16001

[11] A. M. Turing. 1948. Turing, Intelligent Machinery, A Heretical Theory. *Alan M. Turing* (1948), 128–134.

[12] H. van Hasselt and M. A. Wiering. 2007. Reinforcement Learning in Continuous Action Spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. 272–279. https://doi.org/10.1109/ADPRL.2007.368199

[13] C. J. Watkins. 1989. *Learning from Delayed Rewards.* Ph.D. Dissertation. King's College, Cambridge, UK. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

[14] C. J. Watkins and P. Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (01 May 1992), 279–292. https://doi.org/10.1007/BF00992698

[15] J. Zhong, C. Weber, and S. Wermter. 2012. A Predictive Network Architecture for a Robust and Smooth Robot Docking Behavior. *Paladyn* 3 (12 2012), 172–180. https://doi.org/10.2478/s13230-013-0106-8