ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.

Neural Fitted Actor-Critic

Matthieu Zimmer[(1)], Yann Boniface[(1)], and Alain Dutech[(2)] *

(1) University of Lorraine, LORIA, UMR 7503, F-54000, Nancy, France
(2) INRIA, LORIA, UMR 7503, F-54000, Nancy, France

Abstract. A novel reinforcement learning algorithm that deals with both continuous state and action spaces is proposed. Domain knowledge requirements are kept minimal by using non-linear estimators and since the algorithm does not need prior trajectories or known goal states. The new actor-critic algorithm is on-policy, offline and model-free. It considers discrete time, stationary policies, and maximizes the discounted sum of rewards. Experimental results on two common environments, showing the good performance of the proposed algorithm, are presented.

1 Introduction

Reinforcement learning (RL) [1] is a framework for solving sequential decision problems where an agent interacts with its environment and adapts its policy based on a scalar reward signal. RL agents can autonomously learn difficult tasks, like navigating a maze or playing a video game [2]. While the basic setting of RL is now well established, a number of researchers have been studying variants where environments with continuous spaces lead to more and more practical problems.

Thus, the purpose of this article is to present an RL algorithm respecting two main requirements : 1) dealing with continuous state and action spaces in order to address more realistic problems, 2) the knowledge added by the designer to the agent should be minimal, to allow the agent to acquire progressively its own representations in a developmental robotics perspective [3].

Firstly, the common RL background of classical algorithms and their limitations are described. Secondly, the main algorithm of this paper is exposed with an experimental comparison on multiple environments.

2 Background

2.1 Reinforcement learning

RL is a framework that models sequential decision problems, in which an agent learns to take better decisions while interacting with its environment. Once an agent performs an action, the state changes and the agent receives a scalar value, which can be null, called reward, that encodes information about the quality of the actual transition. The goal of the agent is to maximize its long-term expected total reward.

*The data has been numerically analyzed with the free software package GNU Octave [15]. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

The underlying formalism of RL is that of *Markov Decision Processes* (MDP). An MDP is formally defined as a tuple $\langle S, A, T, R \rangle$ where S is a set of states, A a set of actions, $T : S \times A \times S \to [0, 1]$ are transition probabilities between states ($T(s, a, s') = p(s'|a, s)$ is the probability of reaching state s' from state s after executing action a) and $R : S \times A \to \mathbb{R}$ is a reward signal. A *policy* $\pi : S \times A \to [0, 1]$ encodes how the agent will behave (the probability to take an action in a given state). An *optimal* policy π^* maximizes the expected discounted reward, that is :

$$\pi^* = \arg\max_{\pi} J(\pi) = \arg\max_{\pi} \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t \times R(s_t, \pi_t(s_t))\Big] \tag{1}$$

where t denotes a time step and $0 < \gamma < 1$ is a discount factor. In the RL setting, one tries to learn an optimal policy when the model, T and R, is unknown. Such a setting is considered in this paper.

2.2 Dealing with continuous state space

When the space S is continuous, classical value-function based RL methods like Least-Squares Temporal Difference (LSTD) [4] rely on an estimation of $Q : S \times A \to \mathbb{R}$, the (sequential) values of actions in each state, or $V : S \to \mathbb{R}$ the value of each state.

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}\Big[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \Big| s_t = s, a_t = a\Big] \tag{2}$$

This estimation is usually a linear combination of provided basis functions, that must be carefully defined by an expert. Their nature and number limits the expressivity of the learned function. It also impedes the agent's capacity to develop its own representations. In order to minimize the knowledge brought by experts (requirement 2), the use of non-linear functions (like neural networks) will be explored as they have a greater expressiveness range and avoid the *a priori* definition of basis functions.

2.3 Dealing with continuous action space

Critic only : Critic-only methods derive a policy directly from the value function :

$$\pi(s) = M(Q, s, A) = \arg\max_{a \in A} Q(s, a) \tag{3}$$

where the function M can be the arg max operator, Boltzmann soft max, etc. Several algorithms [1] have been designed with different update rules : Q-Learning, Sarsa, LSTD, etc. Among these, Fitted Q Iteration (FQI) [5] is particularly interesting as it uses non-linear neural networks, is data efficient and takes advantage of the Rprop [6, 7] backpropagation algorithm.

The main issue with critic-only methods is that they are very difficult to apply to *continuous action* spaces. The argmax operator cannot be used and

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.

optimizing the Q function, a linear combination of non-linear basis functions, is a complex task.

Actor-only : On the other side, it is possible to define a parametric policy π_θ without value function [8]. The goal is to find the best parameters θ according to the cost function $J(\pi_\theta)$ from (1) by exploring the finite-dimensional parameter space Θ. Evolutionary algorithms like Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [9] or gradient descent are common optimization tools. The major drawbacks of actor-only methods are the high variability of the cost J and, for gradient-based methods, the plateau effect and local minima that can lead to poor policies [10, 11].

Actor Critic : Actor-critic algorithms [11] try to combine the advantage of both previous methods. The critic learns the value function to reduce the variability of the approximation of J and the actor learns the policy allowing the use of continuous actions. Continuous Actor Critic Learning Automaton (CACLA) is a successful actor-critic algorithm [12] that uses neural networks for both the critic and the actor. However, it is an online algorithm, as such is not *data efficient* as every experienced interaction is only used once.

3 Neural Fitted Actor-Critic

Neural Fitted Actor-Critic (NFAC) is a novel actor-critic algorithm whose roots lie in both CACLA and Q-Fitted Iteration. It is an *offline actor-critic* algorithm that can re-use data and deal with continuous actions. It proceeds in two steps summarized in Figure 1 : 1) given the current policy π, it samples and collects interaction data into $\mathcal{D}_\pi$, 2) improves the critic approximation and actor policy with $\mathcal{D}_\pi$.

Interaction data collection D_π is made of several tuple of $(s_t, u_t, a_t, r_{t+1}, s_{t+1})$. For a state s_t, the actor provides the current best action u_t, which is slightly altered in an exploration action a_t (sometimes equals to u_t), which leads to a new state s_{t+1} and reward r_{t+1}.

Neural network estimation The actor and the critic are implemented using multi-layer neural networks. Their update phases which are detailed below make use of Rprop back-propagation algorithm [6, 7] which avoids the definition of learning rates.

3.1 Critic update

The critic's update relies on several *supervised* learning iterations as done in FQI. At each iteration k, a learning base $\{(s_t, v_{k,t})\}$ of $(S \times \mathbb{R})^{|\mathcal{D}_\pi|}$ is built from $\mathcal{D}_\pi$ using the current approximation of V_k. Each state s_t of $\mathcal{D}_\pi$ is associated a target $v_{k,t}$ which is either $r_{t+1} + \gamma V_k(s_{t+1})$ or r_{t+1} when s_{t+1} is a goal state or absorbing state. Thus V_{k+1} is computed as follows :

$$V_{k+1} = \underset{V \in \mathcal{F}_c}{\operatorname{argmin}} \sum_{s_t \in \mathcal{D}_\pi} \Big[V(s_t) - v_{k,t} \Big]^2 \tag{4}$$

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

$\mathcal{F}_c$ is the search space of the critic : a subset of learnable functions (for instance a multi-layer perceptrons). The learning base must be rebuilt at each iteration since the targets $\{v_{k,t}\}$ to learn V_{k+1} are dependent on V_k. Unlike FQI which is *off-policy*, and because of the actor-critic setting, V is here an *on-policy* evaluation of the current actor. This implies that D_π must be rebuilt after each modification of the actor.

3.2 Actor update

The actor update modifies the policy in order to reinforce exploratory actions a_t that do better than the current policy actions u_t. The actor update relies on the temporal difference error $\delta_t = \big(r_{t+1} + \gamma V_k(s_{t+1})\big) - V(s_t)$. Since δ_t depends on the approximation of V, the actor update is accomplished before the critic update. Like CACLA algorithm, the update is performed towards the exploratory action a_t only when $\delta_t > 0$. However, the process is not online but done according to $\mathcal{D}_\pi$. A unique database $S \times A$ is enough : there is no incremental process as was required with the critic because there isn't a self-dependency in the update formula :

$$\hat{\pi}^* = \underset{\pi \in \mathcal{F}_a}{\operatorname{argmin}} \sum_{(s_t,u_t,a_t)\in\mathcal{D}_\pi} \begin{cases} \big(\pi(s_t) - a_t\big)^2, & \text{if } \delta_t > 0 \\ \big(\pi(s_t) - u_t\big)^2, & \text{otherwise} \end{cases} \tag{5}$$

Notice that an exploitation case, where a_t is equals to u_t, could also result in a policy update. Whereas CACLA is online, and can only reinforce an exploration action, here the offline setting of NFAC allows modifying more deeply the policy by trying to reproduce this exploratory action.

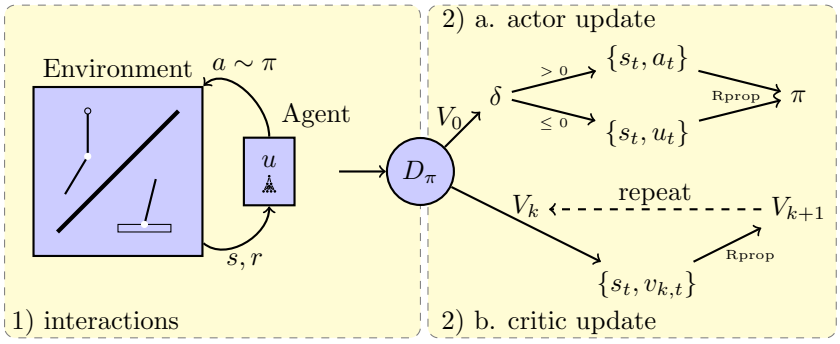

Fig. 1: Flow diagram of NFAC algorithm.

4 Experimental Setup

NFAC and CACLA are compared in two continuous environments : Acrobot [13] and Cartpole [14]. Acrobot (double swing-up) is an under actuated arm of two-link. The first joint cannot exert a torque where the second can. It always

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8.
Available from http://www.i6doc.com/en/.

begins in the same position (straight down). The reward function is defined as 1) +1 if the goal is reached, 2) normalized max height of end effector if 500 steps are reached, 3) 0 otherwise.

Cartpole or inverted pendulum is also made of two joints. The first joint is a pivot point between the cart and the pole, where torques cannot be applied. The other joint controls the horizontal movement of the cart. The reward function is defined as 1) -1 when the pole touches the ground 2) +1 otherwise.

Simulations have been done with the Open Dynamics Engine (ODE) [15] with the following parameters (soft_erp : 0.05, soft_cfm : 0.95) :

	Mass	Gravity	Torque	Time Step
Units	g	$m \cdot s^{-2}$	N	s
Acrobot	127, 127	9.81	$[-0.125; 0.125]$	0.01
Cartpole	85, 38	9.81	$[-5; 5]$	0.01

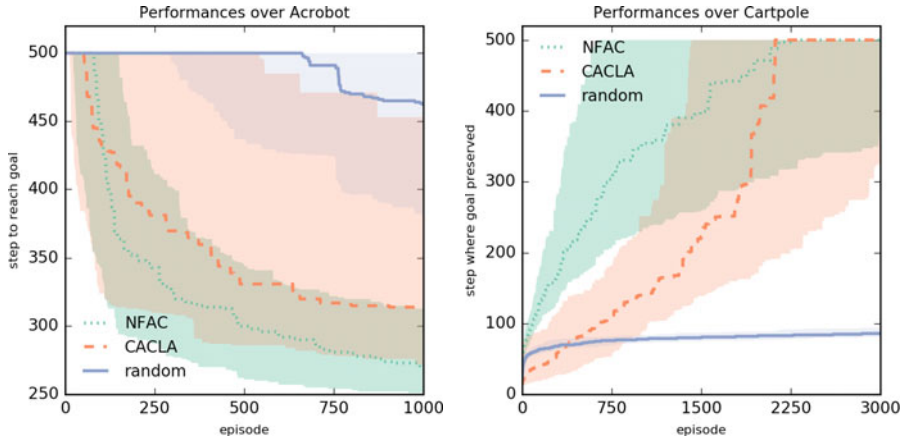

Fig. 2: Median and quartile of the best registered performance in Acrobot (lower better) and Cartpole (higher better) environment during RL learning.

As seen in the Figure 2, NFAC has better performance than CACLA. In acrobot, 75% of NFAC agents reach their goal after only 151 episodes where CACLA agents need 544 episodes. The median shows that the quality of the policies found by NFAC are better. After 1000 episodes, 75 % of NFAC agents can reach the goal after only 313 steps versus 453 steps for CACLA.

In cartpole, 25% of NFAC agents can preserve the goal during 500 steps after 623 episodes versus 1419 episodes for CACLA. The median of NFAC is better than CACLA during the first episodes then converge after 2200 episodes.

Statistics have been made over 150 different runs after meta-parameters optimization for each algorithms (the number of hidden units for the policy is the same : 5). ϵ-greedy policies are used with acrobot, and Gaussian policies with cartpole. In some environments CACLA has shown better than CMA-ES and Natural Actor-Critic [16].

ESANN 2016 proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. Bruges (Belgium), 27-29 April 2016, i6doc.com publ., ISBN 978-287587027-8. Available from http://www.i6doc.com/en/.

5 Conclusions and further work

NFAC is an actor-critic algorithm that considers continuous spaces, using non-linear estimators for both the actor and the critic. It is inspired by CACLA and FQI. Unlike FQI, NFAC can be used in environment with continuous actions. Additionally, conducted experiments showed that NFAC performs better than CACLA with fewer meta-parameters. To further improve NFAC, data collected in previous episodes should be re-used, for instance by using an importance sampling algorithm where the difficulty lies in identifying pertinent data. This algorithm is also a first step towards deep reinforcement learning with continuous action spaces, since it would be able to use neural networks. Additionally, it could be used in a developmental robotic perspective since it requires very little predefined knowledge.

References

[1] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. A Bradford Book, 1998.

[2] Volodymyr EzaazeMnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Others. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.

[3] Minoru Asada, Koh Hosoda, Yasuo Kuniyoshi, Hiroshi Ishiguro, Toshio Inui, Yuichiro Yoshikawa, Masaki Ogino, and Chisato Yoshida. Cognitive Developmental Robotics : A Survey. *IEEE Transactions on Autonomous Mental Development*, 1(1):1–44, 2009.

[4] Steven J. Bradtke, Andrew G. Barto, and Pack Kaelbling. Linear least-squares algorithms for temporal difference learning, 1996.

[5] Martin Riedmiller. Neural fitted Q iteration - First experiences with a data efficient neural Reinforcement Learning method. In *Lecture Notes in Computer Science*, volume 3720 LNAI, pages 317–328, 2005.

[6] Christian Igel and Michael Hüsken. Improving the Rprop learning algorithm. In *International Symposium on Neural Computation*, pages 115–121, 2000.

[7] Martin Riedmiller and Heinrich Braun. RPROP - A Fast Adaptive Learning Algorithm. In *International Symposium on Computer and Information Science VII*, 1992.

[8] Richard S. Sutton, David Mcallester, Satinder Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *In Advances in Neural Information Processing Systems 12*, pages 1057–1063, 1999.

[9] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

[10] Ivo Grondman, Lucian Buşoniu, Gabriel AD. Lopes, and Robert Babuška. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man and Cybernetics*, 42(6):1291–1307, 2012.

[11] Vijay R. Konda and John N. Tsitsiklis. Actor-Critic Algorithms. *Neural Information Processing Systems*, 13:1008–1014, 1999.

[12] Hado Van Hasselt and Marco A. Wiering. Reinforcement learning in continuous action spaces. In *Proceedings of the IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279, 2007.

[13] Mark W. Spong. Swing up control problem for the acrobot. *IEEE Control Systems Magazine*, 15(1):49–55, 1995.

[14] Martin Riedmiller, Jan Peters, and Stefan Schaal. Evaluation of policy gradient methods and variants on the cart-pole benchmark. *Proceedings of the 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 254–261, 2007.

[15] Russell Smith. Open dynamics engine. 2005.

[16] Hado Van Hasselt. Reinforcement Learning in Continuous State and Action Spaces. In *Reinforcement Learning*, pages 207–251. Springer Berlin Heidelberg, 2012.