Reinforcement Learning through Supervision for Autonomous Agents

Brahim Boulebtateche, Adel Djellal and Mouldi Bedda

Abstract

Reinforcement Learning (RL) is a class of model-free learning control methods that can solve Markov Decision Process (MDP) problems. However, one difficulty for the application of RL control is its slow convergence, especially in MDPs with continuous state space. In this paper, a modified structure of RL is proposed to accelerate reinforcement learning control. This approach combines supervision technique with the standard Q-learning algorithm of reinforcement learning. The a priori information is provided to the RL learning agent by a direct integration of a human operator commands (a.k.a. human advices) or by an optimal LQ-controller, indicating preferred actions in some particular situations. It is shown that the convergence speed of the supervised RL agent is greatly improved compared to the conventional Q-Learning algorithm.
Simulation work and results on the cart-pole balancing problem and learning navigation tasks in unknown grid world with obstacles are given to illustrate the efficiency of the proposed method.

Keywords: Supervised Reinforcement Learning, Autonomous Agents, LQ-controller, Machine Learning.

1. Introduction

Reinforcement Learning (RL) is a general framework in which an autonomous agent tries to learn an optimal policy of actions from direct interaction with the surrounding environment. The reinforcement learning agent learns its environment through trial-an-error interactions [1]. For each action it executes the environment returns a reward indicating how appropriate the action was in the given situation. This paradigm is well suited for learning on many domains where it is inappropriate to specify in an explicit way how to perform a task, e.g. navigating in unknown environment. The agent explores its environment by executing some actions through trial-and-error interactions. After each action, it receives from the environment a scalar signal called reinforcement (reward / punishment) signal that inform on the appropriateness of taking a particular action in a given state. The reinforcement can be positive (reward), negative (punishment), or zero. The goal of RL is to construct an optimal policy of actions for the agent to follow based on observed interactions with the environment. The agent is, thus, trained so that the long-term return of the expected sum of instantaneous reinforcement rewards is maximized.
However, a fundamental problem of standard RL algorithms is that of the curse of dimensionality. Although, many tasks defined over a finite state space can be dealt with successfully in this framework, in real applications, it would take an enormous amount of time for these algorithms to converge towards a suitable solution even for moderately complex state space. There are two major approaches that address the problem of slow convergence in large finite state space or that try to find solutions to problems that seem intractable in complex environments with infinite set of states. The first approach is to apply generalization techniques, which involve approximations of the value function or some tiling of the state space [1, 2]. The second approach is to provide the agent with a priori information about the environment. We can incorporate such a knowledge either by modifying the reward function as in the reward shaping techniques [3] or we can create macro-actions from primitive ones as in [4]. Shaping can be used to speed-up the learning process for a problem or in general to help the reinforcement learning technique to scale to large and more complex problems. To use shaping in practice one must know more about the problem at hand in order to modify the reward function during the learning phase. This may introduce the risk that the agent learns a solution to a problem that is only locally optimal. A macro-action is a way of grouping primitive actions into a new one. For example, if the primitive action is walk one step in a given direction, a macro-action would be to walk some steps to one direction followed by some other steps to another direction. Macro-actions represent the problem at different levels of abstractions. It has been shown that given the right set of macro-actions, a reinforcement learning agent can increase its learning rate drastically [5]. Many methods have been proposed to automate combining primitive actions into macro ones [6, 7]. However, the way these macros are created has a great influence on the final performance of the learning process. A third approach, based on the use of supervision techniques, has been followed by a number of researchers to overcome some of

the difficulties that may arise in previous methods. For instance, many techniques that combine the two concepts of supervised learning and reinforcement learning are well established in robotics and social sciences. Among them, we may cite imitation learning, LQ controller induction, and learning by demonstration, [8] [9] [10]. Despite the many successful implementations, none of these methods combines both kinds of learning instantaneously. Either supervised learning precedes RL during a separate training phase, or else the supervisory information is used to modify a value function rather than a policy.

In this paper, we propose a supervised approach to the classical Q-Learning algorithm of the RL paradigm [11, 12]. The structure of the learning problem is modified by incorporating some a priori knowledge provided by a supervisor during the training phases. In this paper, we present a supervised RL scheme that intends to improve learning in complex environments. With this approach, a supervisor adds structure to the classical framework for RL by integrating on-line a priori information or advices to the learning agent and that may help speeding-up learning tasks. Two examples are described to illustrate ideas behind our algorithm by using two types of supervisors: a human operator that issues intermittent advices to guide an autonomous agent learning navigation in unknown environment with obstacles, and a feedback LQ controller that is easily designed yet sub-optimal to help selecting appropriate actions for balancing a nonlinear system (an inverted pendulum on a moving cart). The remaining part of the paper is organized as follows: Section 2 provides a succinct review of the RL principles used to solve typical class of problems represented by MDPs models. Section 3 focuses on the use of supervisory techniques as applied to RL. This structure seeks to improve the learning rate by incorporating added knowledge to Q-learning algorithm during the training process, i.e., on-line. In sections 4 and 5 are described in more details the insertion of two different supervisors into the RL structure for two domains: one is learning a navigation task in a grid world with obstacles and the other is balancing an inverted pendulum on a cart. Finally, a conclusion is drawn in section 6 and avenues for future work are identified, therein, as well.

2. Brief Review of Reinforcement Learning

RL is the problem of learning an optimal behavior from direct interaction with an environment. Following is a succinct account of a reinforcement learning framework described from a Markov Decision Process (MDP) perspective.

The interaction between the learner and the environment having a MDP structure can be fully described by a finite set of states S , a finite set of actions A , and a real valued reward function, r (s ,a) : $S \times A \rightarrow R$. At some discrete time step $t \in T$ the learner is in some state $s_t \in S$ where it can choose to perform an action $a_t \in A_s$, where $A_s \subset A$ is the set of available actions in state s. Upon execution of action a_t , it may result a state transition whereby the learner will find itself in state s_{t+1} with probability P(s_t , a_t , s_{t+1}), which is referred to as the transition probability. Arriving in state s_{t+1} the learner receives a reward r_t (s_t ,a_t) from the environment . Whereas the reward function gives an indication of the immediate utility of taking action a in state s and then following some policy. A policy π is defined as a mapping from states to actions. An optimal policy is a policy that optimizes some function of reward (either maximizing gain or minimizing cost) in the long run.. Furthermore, it is helpful to define a real valued function Q (s, a): $S \times A \rightarrow R$, named the action-value function [11, 12]. The goal of RL is to derive the optimal action-value function, Q^* (s , a) from these interactions, for taking action a in state s and terminating in state s':

$$Q^*(s , a) = E\{r (s ,a) + \lambda \max_{a'} Q^*(s', a') \} \qquad (1)$$

where $\lambda \in [0 , 1]$is the discount factor, and E is the expectation operator. An iterative version of the optimal action-value function is given by the Q-Learning algorithm [ ]. All state-action pairs are stored in a table, and their update takes place based on experiences (s , a , r , s') according to :

$$Q(s , a) \leftarrow \alpha (r (s , a) + \lambda \max_{a'} Q (s', a')) + (1 - \alpha) Q (s , a) \qquad (2)$$

where α is the learning rate and gives a trade-off between a new observation and the present approximation . When the convergence of the Q-Learning algorithm is reached, the policy that it defines (referred to as the greedy policy) is simply obtained by taking actions with maximum value for the current state, given by :

$$\pi (s) = \operatorname{argmax}_a Q(s , a) \qquad (3)$$

However, there are some problems associated with using Q-Learning on complex environments defined by infinite state/action space. Learning how to act in such domains is not guaranteed to converge to an optimal policy. Therefore, a modified structure of RL is well needed in order to effectively adapt to complex problems. As stated earlier, among many techniques investigated to alleviate the RL deficiencies, methods that, somehow, try to combine supervisory information and RL form a natural trend in machine learning.

This will be considered with more details in the next section.

3. Supervised Reinforcement Learning

Most work on reinforcement learning has concentrated on the tabular case, in which the agent has little or no prior knowledge when it begins to act and learn in a finite state space. It is well known that in all branches of machine learning that, without some significant bias (or prior knowledge), learning cannot be efficient or effective. In reinforcement learning, the problem is even worse: if an agent starts with no knowledge at all and begins to act at random, it may take an extremely long time for the agent to even encounter the parts of its environment from which it can learn. For this reason, we must find ways to introduce additional knowledge into reinforcement learning agents.

One of the most appealing methods, which is widely used in nature, is for an agent to learn by watching and imitating other agents. This is a very practical way for an agent to gain additional knowledge and it has been the basis for some very successful robot learning programs. However, perception remains a big difficulty; most robots do not have sufficiently advanced perception to be able to sense what other robots are doing and whether they are succeeding.

The least attractive, but perhaps most immediately practical, method is for prior knowledge to be given directly to the robot. This knowledge might be in the form of partial or incorrect programs, program decompositions, or local reinforcement function. This initial knowledge, even if it is partial or sub-optimal, may guide the robot to behave well enough initially to learn effectively from its environment.

Supervised reinforcement learning as another method for improving the effectiveness of learning is a rule rather than the exception in natural life. With this approach, a supervisor adds structure to a learning problem and supervised learning makes that structure part of the reinforcement learning framework. The supervisory part of this structure could be of two kinds : a feedback controller that is easily designed yet sub-optimal, and a human operator providing intermittent command or advices to an autonomous learning agent. The supervisor agent or trainer may intervene at different levels of abstraction during the learning process: at state level by identifying some special states as sub-goals [7] , at reward level by modifying the reward function as in reward shaping techniques, or at a more abstracted level by providing macro-actions or partial behaviors to the learning agent. Figure 1 illustrates the general idea behind the supervised RL paradigm. Initially, the learning agent starts exploring the environment by executing some actions based on trial-and-error interaction. After each action taken, it receives from the environment a scalar signal that indicates the value of that action with respect to the given task goal to be accomplished. To speed-up the learning phase and reducing the exploration time needed to learn about the environment, a supervisor is used to provide the learning agent with pertinent information. Thus, the learning agent is instructed to take some privileged actions in particular situations, as specified by the supervising agent. In doing so, integrating a priori information into the learning agent policy leads to effectively directing search during learning and allowing relatively quick convergence to successful control policies.

A key feature of reinforcement learning is the exploration/exploitation trade-off. To accumulate as much reward as possible, a learner must exploit the knowledge it already has. However, some actions with small immediate reward may yield even more reward in the long run, but to find out about them the learner has to choose them, even though they do not look promising. Therefore, the choice of actions, between exploration or exploitation, can have significant effect on the behavior of the learner.

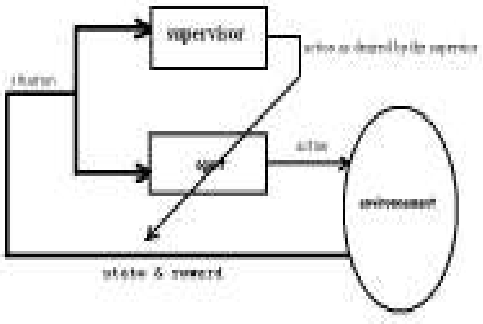

Figure 1. Supervised RL structure

In our approach, the supervised reinforcement learning is implemented by indicating to the RL agent, through a human operator, some interesting parts of the environment state space , known as 'way-point ' states, in which the learning agent is instructed to perform an appropriate given action for navigation purposes . Whenever the agent reaches one of these states it breaks down its usual way of selecting actions (for instance, ε- greedy policy is commonly used) and proceed its search by following related actions fixed by the supervisor agent. Another method that we also implemented, here, is based on a direct training of the learning agent by an LQ-controller (Linear Quadratic controller) used a teacher. The learning agent try to imitate direct actions of the controller by passively observing the control signal of the latter. Thus, the agent's action policy follows, loosely speaking, the same trend as that of the controller. The reason for

choosing this form of supervision is that incorporating the trainer's action, somehow, directly into the learning agent's building policy will not affect the underlying structure of the reinforcement learning process. Furthermore, due to the generalization capability of the RL, it is believed that the RL agent will well generalize to a wide range of conditions than the LQ-controller which can only perform robustly under limited conditions (i.e. , constrained to work only near around the equilibrium state of the linearized model of the pendulum-cart system). The conventional RL is slightly modified at the level of the search procedure in the action space: it uses ε-greedy policy, i.e. an action leading to the best estimate of Q(s , a) function is chosen most of the time (with a probability equal to 1 - ε). However, a small fraction of the time, ε , an action is given by the LQ-controller instead of being selected randomly from the action space. The following section describes in more detail the application domains used to implement and illustrate the effectiveness of the proposed approaches.

4. Experimental work and Results

The objective of these experiments is to compare the performance of supervised RL as implemented by our methods with that of conventional Q-Learning algorithm. The two standard problems we used for this purpose are often utilized for testing learning algorithms performance: the navigation learning problem of an autonomous agent and the balancing problem of an inverted pendulum on a moving cart.

4.1 Navigation task problem

Our first experiment demonstrates the integration of intermittent user advices into an autonomously navigating agent. The goal of the agent navigation task is to learn to navigate in the unknown environment and reach a specific target state in an optimal manner. The environment, as shown in Figure 2, is presented as a grid world of 10 × 10 square cells (where free cells are in yellow color and obstacle cells or barriers in red). A state of this environment is indicated by the location of a cell. A set of waypoints are superimposed at particular location on the grid. These waypoint states represent user a priori information that gives hints to the navigating agent. The action space is composed of four actions: the agent has to move in one of the four directions, namely, up, down, right and left direction. The latter receives a reward of – 1 whenever it takes an action that makes a transition to a free state. An attempt to move into a wall or an obstacle is given a penalizing reward of -100. For instance, the choice of actions, directed or undirected, has a significant impact on the behavior of the agent. Initially, the autonomous agent is in a start position, referred to as the start state, and seeks to attain a final state known as the goal state by learning to avoid obstacles by its own. A waypoint state is used to reduce the learning time by directing exploration toward the goal state, almost exactly, in the same way a tourist does when he or she is exploring an unknown town with a guide. At any given waypoint, the agent is forced locally to follow a preferred action regardless of any other action that may be available in that state. However, in all other situations, the agent applies an ε- greedy policy of action selection. In this experiment, the whole system (learning agent and grid-world environment) was implemented in Matlab. The Q-Learning algorithm was slightly modified by inserting a priori knowledge in its structure. A discount factor , λ , equal to 0.95 was used, and the ε- greedy policy was selected with a probability of 1 - ε (where ε = 0.1). Given that the state space is relatively small, we used a table-based representation to store the Q-values. Figures 3 and 4 show the results obtained by our method and by the standard Q-Learning algorithm. It is, clearly, seen on both figures that the supervised RL approach has better ability to learn the navigation task than the unsupervised version of the Q-learning method. The curves show that the supervised agent needed less time to learn a good policy: the number of episodes required for the supervised agent to find the optimal path to the goal is 20 whereas for the unsupervised agent it is 30. We, also, observe that the number of steps to goal, obtained by our algorithm, is significantly less than that realized by the standard Q-Learning. In conclusion, w may say that the learning speed and the number of succeeded steps to goal are relatively improved by using a priori knowledge.

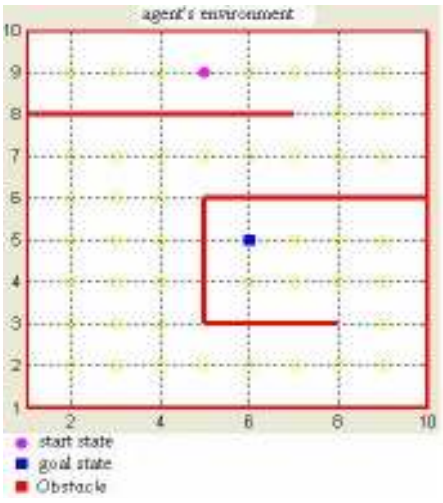

Figure 2. Test environment for the navigation learning task

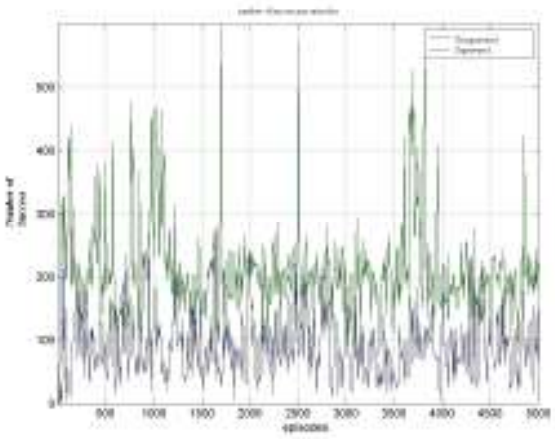

Figure 3 . Reinforcement Learning Performance with supervision (green) and without (blue) for the navigation learning task.

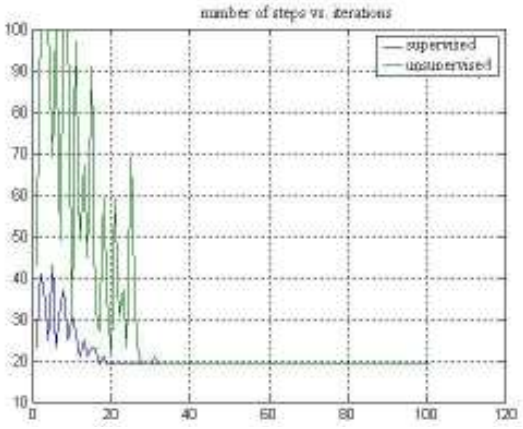

Figure 4. Learning curves: Steps to converge to optimal learned policy

4.2 Inverted Pendulum Balancing Task

In this experiment, we apply our algorithm to the standard cart-pole learning task, a.k.a. inverted pendulum problem, which involves learning to stabilize the upward equilibrium state of the inverted pendulum mounted on a mobile cart by applying appropriate forces to the cart, see Figure 5. The moving cart carries a pendulum that can swing freely around its origin. This system is described by the following nonlinear dynamic state equation:

$$\ddot{\theta} = \frac{g.\sin\theta + \cos\theta\left[\dfrac{-F - m_p.l.\dot{\theta}^2.\sin\theta + \mu_c.sig(\dot{x})}{m_c + m_p}\right] - \dfrac{\mu_p.\dot{\theta}}{m_p.l}}{l\left[\dfrac{4}{3} - \dfrac{m_p.\cos^2\theta}{m_c + m_p}\right]}$$

$$\ddot{x} = \frac{F + m_p.l.\left[\dot{\theta}^2.\sin\theta - \ddot{\theta}.\cos\theta\right] - \mu_c.sig(\dot{x})}{m_c + m_p}$$

The position and velocity of the cart (x , $\dot{x}$) and the pole angle and angular velocity (θ , $\dot{\theta}$) represent the state of the cart-pole system. The specification for the simulated pole and cart are as follows: length of the track, L = 4.0 m ; mass of the cart , m_c = 1.0 kg ; mass of the pole : m_p = 0.1 [kg] ; half length of the pole : l = 0.5 [m] ; gravity : g = 9.8 [m/s2]; coefficient of friction of cart on track : μ_c = 0.0005 ; coefficient of friction of pendulum on cart : μ_p = 0.000002

Only three actions F = {- 10 N, 0, + 10 N} were available to the RL agent for this task.

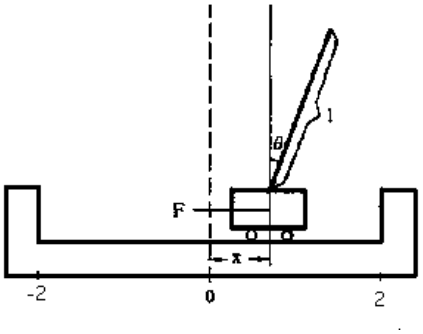

Figure 5. The cart-pole system

Learning to control such a nonlinear system to achieve a certain behavior is not an easy task. The proposed algorithm uses an LQ-Controller for supervisory purposes. This controller is applied to the linearized model of the cart-pole system by using Euler approximation technique with a time step of 0.02 s and intervene only during the balancing task of the pendulum (for $|\theta| \leq 15°$) . The state space is partitioned into boxes of equivalent states (12705 boxes were used for the swinging up task and 270 boxes for the balancing task).

The whole system (RL agent , pole-cart model and LQ-controller) was implemented in Matlab. The Q-Learning algorithm utilizes the same parameters as indicted in the first experiment. At the beginning of each episode, the pendulum is initialized in downward position (angle $\theta = 180°$) with zero angular velocity. The agent has three available actions corresponding to forces of - 10, 0, +10. These actions are chosen small enough so that the only way to move the pendulum higher and higher is by swinging back and forth. The objective is to find a policy that will drive the pendulum past the upward position (the unstable equilibrium state) and maintaining the pendulum in that position for as long as possible.

As is illustrated by the learning curves depicted in Figures 6 through 9, the supervised RL agent gives better results in terms of convergence speed than those obtained without any supervision, and in terms of robustness to perturbation compared to the LQ-Controller. Figure 6 shows that the learning performance of the supervised agent is higher compared to the unsupervised case. The obtained curves are plotted by averaging produced success during 50000 learning episodes. Results for the pole balancing learning problem are depicted in Figure 7 where it is shown that the proposed agent has more success in maintaining the inverted pendulum in the vertical position within an angle less than 15°. If we looked at the magnitude of command signal that are issued by the RL agent and the LQ-controller we can easily notice on Figure 9 that the latter produce action signals of very high magnitude (much more greater than ± 25 N at the beginning of the balancing task) whereas actions of the former are limited to ± 10 N only. Thus, if the command signal is constrained to acceptable values, i.e. ± 10 N for instance, it will take a longer time for the LQ-controller to stabilize the inverted pole. Figure 8 describes the reaction of the two controllers to an impulsive perturbation (the pole has been given an impulse of 75° / s) . We can notice that the RL-based controller was very rapid in, effectively, responding to this perturbation by driving and stabilizing the pole to its vertical position than it is done by the LQ-controller (it takes 250 iterations for the controller, i.e. $250 \times 0.02s = 5$ s, whereas for the RL agent it is just 30 iterations , i.e. $30 \times 0.02s = 0.6$ s). Due to its generalization property, the RL agent demonstrates a better ability to adapt to strong perturbations.

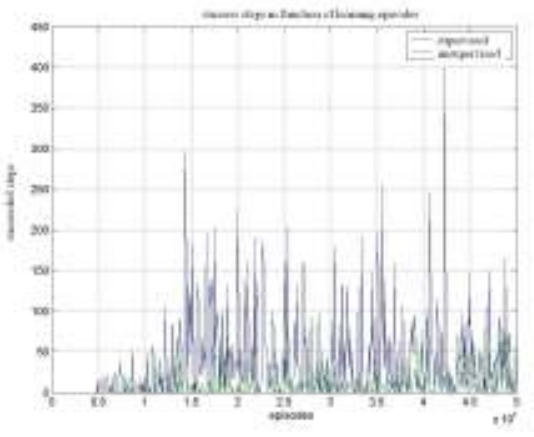

Figure 6. Swing up task. Curves show number of successful steps for swinging up the pendulum versus learning episodes. Supervised (blue curve) and unsupervised RL (green curve)

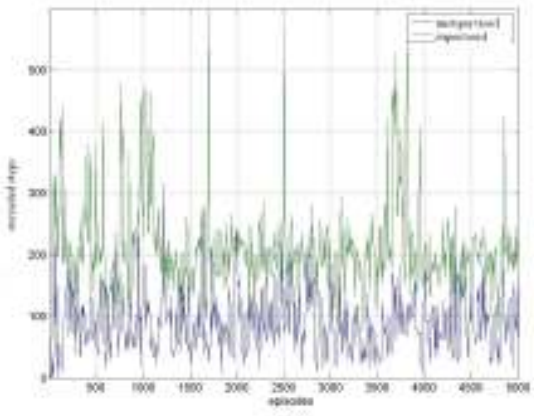

Figure 7. Balancing learning task. Curves show number of successful time steps in balancing the pendulum in function of learning episodes.

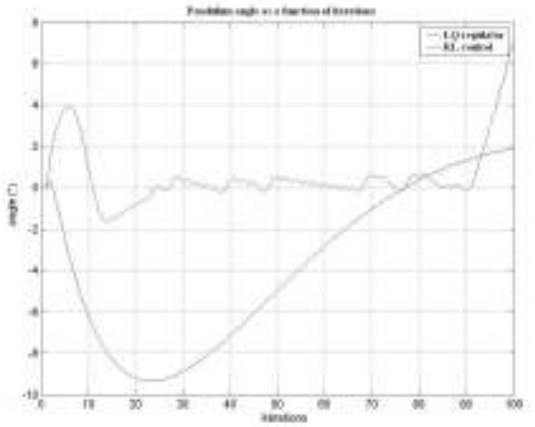

Figure 8. Reaction to a perturbation. RL-based controller (green) and LQ-controller (blue)

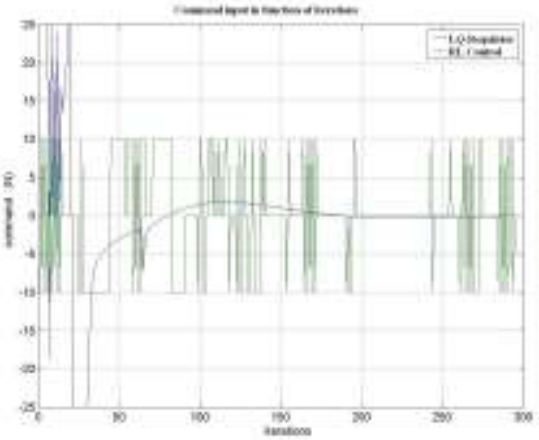

Figure 9. Command signal evolution along iterations, produced during the balancing task by our RL algorithm (green) and LQ-controller (blue)

5. Conclusion

Scaling-up reinforcement learning techniques to complex environments is a challenging task for the AI community. It was our aim here to propose a simple method to accelerate convergence to an optimal policy by using a priori knowledge about the environment to be learned or by following advice proffered by an external controller. This is a modified version of the standard unsupervised RL algorithms built by incorporating supervision within this structure which suffers from low convergence speed in domains with continuous state space. The proposed approach has been implemented on two different environments: one is learning to freely navigate in a grid world containing obstacle and the second task is to learn solving the cart-pole balancing problem. Obtained results are encouraging and, more importantly, they show that supervisory information can really help improving RL performance. However, it should be noted that further work is needed to elucidate all aspects of balancing the need for a priori information integration and the need for sufficient autonomy and exploration by the RL agent in order to not degrade the learning agent's performance.

REFERENCES

[1] Sutton, R. S. and Barto, A. G., Reinforcement Learning : An Introduction. MIT Press, Cambridge, MA, 1998.

[2] Bertsekas, D. P. and Tsitsiklis, J. N. , Neuro-dynamic programming, Athena Scientific, Belmont, MA, 1995.

[3] Dorigo, M. and Colombetti, M. , Précis of " Robot Shaping : An Experiment in Behaviour Engineering" , Adaptive Behavior , 5 (3 – 4)., Précis of the book from MIT Press, Oct. 1997

[6] Dietterich, T. G. , Hierarchical reinforcement learning with the maxQ value function decomposition, Journal of Artificial Intelligence Research, 13 : 227 – 303 , 2000

[4] Mc Govern A. , Sutton, R. S. and Fagg A. H. , Roles of macro-actions in accelerating reinforcement learning, in Proceedings of the 1997 Grace Hopper Celebration of Women in Computing, pages 13 -18, 1997

[5] Precup D. , Temporal abstraction in reinforcement learning, PhD thesis, U. of Massachusetts, Amherst, MA, 2000

[7] Mc Govern A. , Autonomous discovery of temporal abstractions from interaction with an environment, PhD thesis, U. of Massachusetts, Amherst, MA, 2000

[8] Price B. , and Boutilier C. , Accelerating reinforcement learning through implicit imitation, Journal of Artificial Intelligence Research, 19 , pp. 569 – 629 , 2003

[9] Huber M. and Grupen R. A. , A feedback control structure for on-line learning tasks, Robotics and Autonomous Systems, 22 (3 – 4) , pp. 303 - 315 ,1997.

[10] Dixon, K. R. , Malak, R. J. and Khosla, P. K. , Incorporating Prior Knowledge and Previously Learned Information into Reinforcement Learning Agents, Technical Report, Institute for Complex Engineering Systems, CMU, 2000.

[11] Watkins, C. J. C. H. , Learning from Delayed Rewards, PhD thesis, Cambridge University, 1989

[12] Watkins, C. J. C. H. , and Dayan, P. , Q-Learning, Machine Learning, 8, pp. 279 – 292, 1992