Two Steps Reinforcement Learning in Continuous Reinforcement Learning Tasks

Iván López-Bueno, Javier García, and Fernando Fernández

Universidad Carlos III de Madrid
Avda. de la Universidad 30, Madrid, Spain
ivan.lopez-bueno@alumnos.uc3m.es, fjaviergp@gmail.com,
ffernand@inf.uc3m.es

Abstract. Two steps reinforcement learning is a technique that combines an iterative refinement of a Q function estimator that can be used to obtains a state space discretization with classical reinforcement learning algorithms like Q-learning or Sarsa. However, the method requires a discrete reward function that permits learning an approximation of the Q function using classification algorithms. However, many domains have continuous reward functions that could only be tackled by discretizing the rewards. In this paper we propose solutions to this problem using discretization and regression methods. We demonstrate the usefulness of the resulting approach to improve the learning process in the Keepaway domain. We compare the obtained results with other techniques like VQQL and CMAC.

1 Introduction

Two Steps Reinforcement Learning (2SRL) is a method to compute the action-value function in model free Reinforcement Learning [5]. The method assumes a reduced set of actions and finite trials, where positive and discrete rewards can be obtained only when a goal area is achieved. It is based on finding discretizations of the state space that are adapted to the value function being learned, trying to keep the convergence properties of the discretized methods using non-uniform discretizations [11]. In this case, we learn the action value function, $Q(s, a)$, instead of the state value function, $V(s)$, learned in Dynamic Programming [9]. The method is based on two learning phases. The first one is a model free version of the Smooth Value Iteration algorithm [2], that is called Iterative Smooth Q-Learning (ISQL). This algorithm, that executes an iterative supervised learning of the Q function, can be used to obtain a state space discretization too. This new discretization is used in a second learning phase, that is called Multiple Discretization Q-Learning (MDQL), to obtain an improved policy. Performing both phases demonstrated more accurate results in different domain [5].

However, the method presented a main drawback: it requires a discrete reward function. Using such discrete reward function permits ISQL to learn a function approximation of the Q function applying classification algorithms such as J48,

J. Cabestany et al. (Eds.): IWANN 2009, Part I, LNCS 5517, pp. 577–584, 2009.
© Springer-Verlag Berlin Heidelberg 2009

an algorithm to learn decision trees [10]. However, many domains, like Keepaway, have continuous reward functions. In this case, 2SRL needs to discretize the reward space by hand, and to test different discretizations to obtain an accurate one.

To apply the same ideas of 2SRL in domains with continuous rewards requires that the function approximation used in the ISQL phase be a regression approach. As before, the approximation method should generate a discretization of the state space, so such discretization can be used in the second learning phase. Fortunately, there are different approaches in the literature for regression that generate state space discretizations of the input space. Classical ones are M5 [10] or PART [4], which generate regression trees and regression rules respectively.

In this work we adapt the original 2SRL algorithm to deal with domains with continuous rewards. Specifically, we use Keepaway [8], a sub-task of the RoboCup simulator, to perform the evaluation. We compare the new approach with the original algorithm (using discretized rewards), and also with previous approaches like VQQL [3] and CMAC [8], obtaining very compiling results.

This paper is organized as follows. Section 2 describes the algorithm 2SRL in domains with continuous rewards. Section 3 shows the evaluation of the new approach. Last, Section 4 describes the main conclusions.

2 Two Steps Reinforcement Learning in Domains with Continuous Rewards

The use of 2SRL requires, mainly, the adaptation of the Iterative Smooth Q-Learning algorithm (ISQL). ISQL is derived of the Discrete Value Iteration [1], where a function approximator is used instead of the tabular representation of the value function, so the algorithm can be used in the continuous state space case. The algorithm is described in Figure 1. The update equation of the Q function used is the stochastic Q-Learning update equation. The figure shows the adapted version of the original ISQL algorithm to permit continuous rewards.

The algorithm assumes a discrete set of L actions, and hence, it will generate L function approximators, $Q_{a_i}(s)$. It requires a collection of experience tuples, T. Different methods can be applied to perform this exploration phase, from random exploration to human driven exploration [7]. In each iteration, from the initial set of tuples, T, and using the approximators $\hat{Q}_{a_i}^{iter-1}(s)$, $i = 1, \ldots, L$, generated in the previous iteration, the Q-learning update rule for deterministic domains[1] can be used to obtain L training sets, $T_{a_i}^{iter}$, $i = 1, \ldots, L$, with entries of the kind $< s_j, c_j >$ where c_j is the resulting value of applying the Q update function to the training tuple j, whose state is s_j.

In the first iteration, $\hat{Q}_{a_i}^{0}(s)$ are initialized to 0, for $i = 1, \ldots, L$, and all $s \in S$. Thus, when the respective c_j are computed, they depend only on the possible values of the immediate reward, r.

[1] The Q-learning update rule for stochastic domains can be used too.

Iterative Smooth Q-Learning with Regression

- Inputs:
  1. A state space X
  2. A discrete set of L actions, $A = \{a_1, \ldots, a_L\}$
  3. A collection T of N experience tuples of the kind $< s, a_i, s', r >$, where $s \in X$ is a state where action a_i is executed, $s' \in X$ is the next state visited and r is the immediate reward received
- Generate L initial approximators of the action-value function $\hat{Q}^0_{a_i} : X \rightarrow \mathcal{R}$, and initialize them to return 0
- $iter = 1$
- Repeat
  - For all $a_i \in A$, initialize the learning sets $T^{iter}_{a_i} = \emptyset$
  - For j=1 to N, using the j^{th} tuple $< s_j, a_j, s'_j, r_j >$ do
    - $c_j = \alpha c_j + (1 - \alpha) \max_{a_r \in A} \gamma \hat{Q}^{iter-1}_{a_r}(s'_j)$
    - $T^{iter}_{a_j} = T^{iter}_{a_j} \cup \{< s_j, c_j >\}$
  - For each $a_i \in A$, train $\hat{Q}^{iter}_{a_i}$ to approximate the learning set $T^{iter}_{a_i}$
  - $iter = iter + 1$

  Until r_{max} is propagated to the whole domain
- Return $\hat{Q}^{iter-1}_{a_i}$, $\forall a_i \in A$

Fig. 1. *Iterative Smooth Q-Learning* algorithm for domains with continuous rewards

The discretizations obtained in this first phase can be used to tune, in a second phase, the action value function obtained in the previous phase, following a multiple (one per action) discretization based approach. This second phase was called Multiple Discretization Q-Learning (MDQL).

The figure 2 shows how to move from the first scheme to the second one, when an approximation like M5 Rules has been used, and L rule sets have been obtained (one for each action). Then, we transform the *RuleSet* obtained by a M5 model tree/rules into a M5 regression tree/rules called *RuleSet'*. We apply this transformation because the M5 model tree/rules uses a linear regression model on the right part of the rule and we want to obtain a numerical value. If we use the M5 regression tree/rules we get a number of the right part of the rules. That could be more useful in the MDQL phase. The left part of the rules of the approximator *RuleSet' i* will be used as the discretization $D_i(s)$, and the right part of the rules of *RuleSet' i* will be located in column i of the generated Q table, providing an initialization of the Q table in the second learning phase[2]. Each RuleSet' may have a different number of rules, so the number of rows of the table is given by the maximum number of rules of the L approximators, and zero values can be used to complete the columns with less rules.

Once the translation from the first scheme to the second one is done, a new learning phase can be executed using the Q-learning update function shown in equation 1.

$$Q(s_t, a_t) \rightarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \qquad (1)$$

[2] We also can initialize to 0 and not to use the right part of the rules.

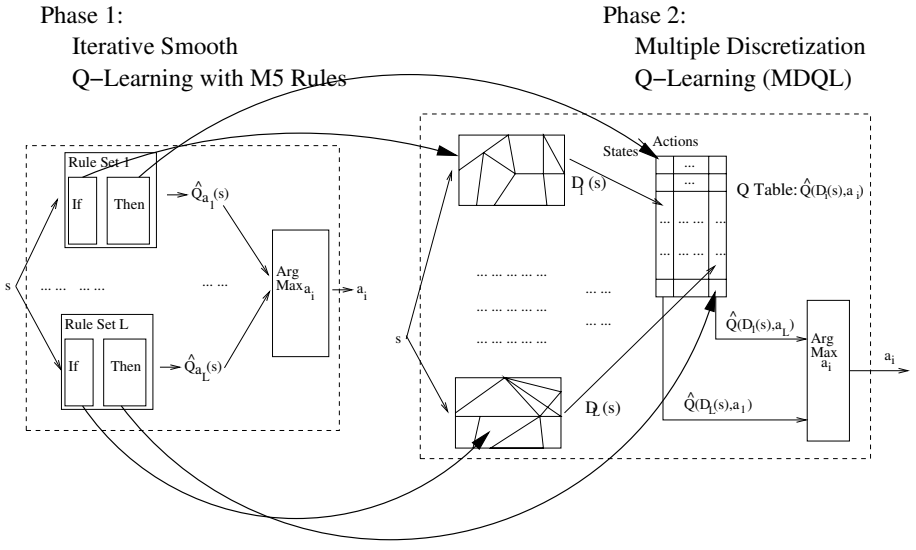

Fig. 2. The two steps of the 2SRL algorithm with Regression

The second learning phase can be executed with new experiences or with the same ones used in the first phase. In the experimentation performed next, the first approach is applied using new experiences. This second phase can be executed exactly as it was defined in the original 2SRL algorithm, so additional explanation of this phase, and how to connect both phases, can be found in the literature [5].

3 Evaluation

To evaluate the new approach, we use the Keepaway subtask of the RoboCup domain [8].

3.1 Keepaway Domain

In this subtask, the keepers try to maintain the possession of the ball within a region, while the takers try to gain the possession. An episode ends when the takers take the possession or the ball leaves the playing region. When an episode ends, the players are reset for another episode. In our experiments each keeper learns independently. From a point of view of a keeper, an episode consists of a sequence of states, actions and rewards:

$$s_0, a_0, r_1, s_1, ..., s_i, a_i, r_{i+1} \tag{2}$$

where a_i is selected based on some perception of s_i. We reward the keepers for each time step which they keep possession, so we set the reward r_i to the

number of time steps that elapsed while following action $a_{i-1} : r_i = t_i - t_{i-1}$. The keepers' goal is to select an action that the remainder of the episode will be as long as possible, and thus maximize the total reward.

The state space is composed of several features that can be considered continuous. These features use information derived from distances and angles between the keepers, takers and the center of the playing area. The number of features used for each state depends on the number of players. In 3vs2 there are 13 features, in 4vs3 there are 19 and in 5vs4 there are 25 features. In the experiments, we use VQ, CMAC and 2SRL as methods for state space generalization in order to improve the learning process. The action space is composed by two different macro-actions, *HoldBall()* and $PassBall(k_i)$ where k_i is the teammate i.

3.2 Results of Two Steps Reinforcement Learning in RoboCup Keepaway

We use both M5 and J48 (C4.5) algorithms to make an approximation of the Q function for each iteration in the ISQL phase. In the case of J48 [10] we have discretized the reward function by hand, and we have applied the original 2SRL algorithm. For M5 [10], we use the adapted version proposed in this paper. The approximator obtained in *ith* iteration is used as a state space generalization in MDQL and the reward tuples are updated with the stochastic Q-Learning update function. The first tuple set was obtained with a random policy working in the keepaway domain.

The parameter setting for 2SRL is the following: $\gamma = 1$, $\alpha = 0.125$. An $\epsilon - greedy$ strategy is followed, increasing the value of epsilon from 0 (random behaviour) to 1 (fully greedy behaviour) by 0.0001 in each episode. The size of the playing region is 25×25. The Q table is initialized to 0. In 3vs2, we obtain the results shown in Figure 3.

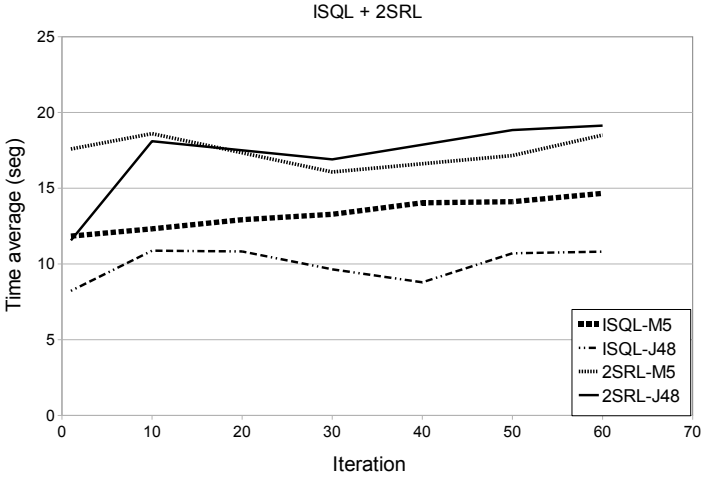

Fig. 3. The two steps of the 2SRL algorithm with M5 and J48

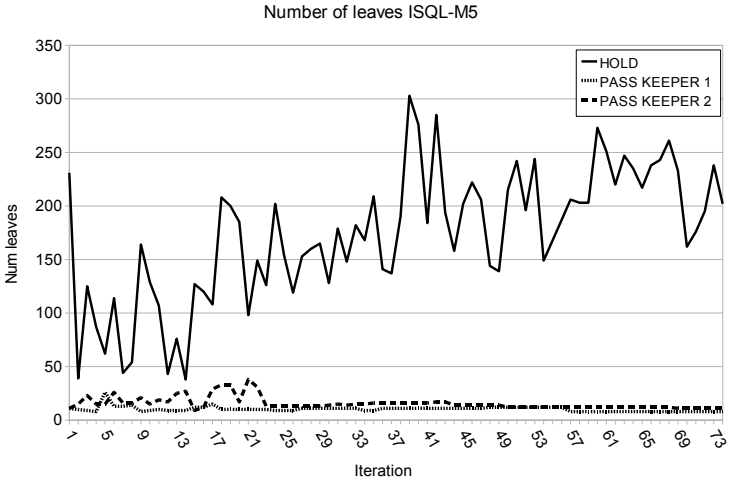

Fig. 4. Evolution of the number of leaves for ISQL-M5

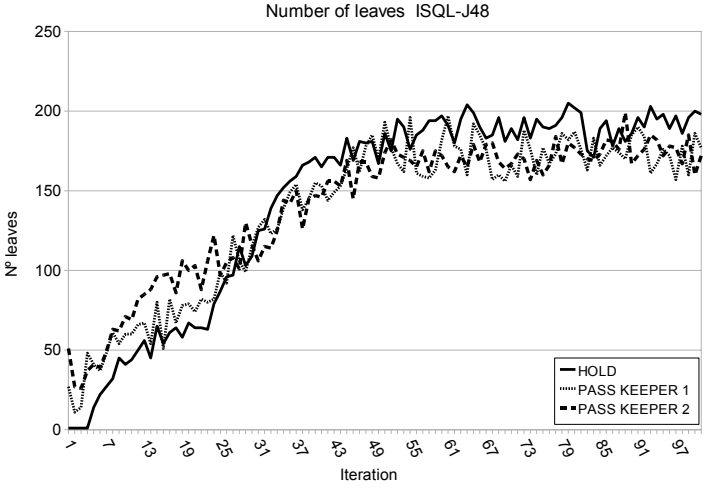

Fig. 5. Evolution of the number of leaves for ISQL-J48

In the graphs, there are four learning curves using ISQL-M5, ISQL-J48, 2SRL-M5 and 2SRL-J48. The *y-axis* is the average episode length when a stable policy is obtained and the *x-axis* is the ISQL iteration. The ISQL-M5 curve was generated using M5 model trees obtaining a very good improvement. The ISQL-J48 curve was generated using the best discretization of the reward space obtained with 88 classes and the performance is worse than M5 model. However, M5 model tree generates a very small state space discretization (Figure 4). For this reason, we transform the *RuleSet* into *RuleSet'* like we explain in section 2.

The trees generated with J48 algorithm have too many leaves ($\approx$ 12000 per action) and the tree leaves are used as space state discretization. We need a smaller discretization more useful for the second phase of 2SRL. We apply different pruning methods to curb the tree growth and to stabilize the space state discretization (Figure 5).

When we use tuned M5 and J48 (88 classes) trees in MDQL we obtain better results than M5 and J48 (88 classes) without tuning. We can see in Figure 3 that the trees obtained from the last iterations of ISQL are more useful in MDQL than the first ones. In the experiments, M5 obtains better or similar results than the best configuration of J48 for ISQL and 2SRL respectively.

3.3 Comparison with Different Approaches

In this section we compare 2SRL with other techniques: the best configurations obtained by hand for VQQL [3] and CMAC [8].

In the experiments presented in this report, 2SRL and VQQL obtain similar results in spite of to be very different techniques. The best VQQL experiment has 64 states and 2SRL has 274(M5) and 553(J48) states. In complex domains with a large state space, the choice of the discretization method may have dramatic effect on the results that we obtain. For this reason, the state representation is chosen with great care. In our experiments we have used ISQL-2SRL to tune the state space discretization obtained from M5 and J48 algorithms. CMAC results [8] are better than 2SRL and VQ results in 3vs2 (Figure 6).

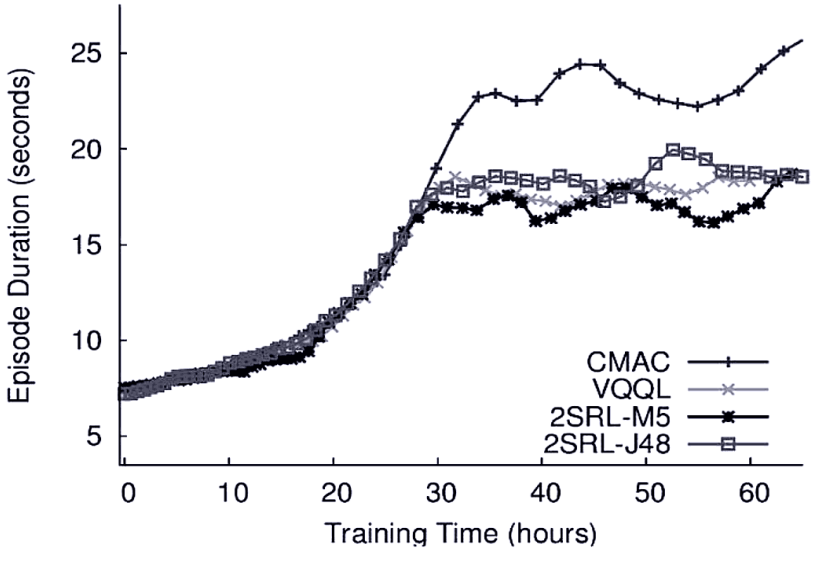

Fig. 6. CMAC, VQQL, 2SRL algorithm with M5 and J48

4 Conclusions

In this paper we have applied two steps reinforcement learning in Keepaway domain. Keepaway domain has continuous rewards and we have adapted the original 2SRL algorithm to deal with domains with continuous rewards. We use M5 and J48 to approximate the Q function in the first phase and we use the same approximators as discretization of the continuous keepaway state space, obtaining very successful results. In the experiments, we present the result of the best J48 discretization obtained after an exhaustive experimentation evaluating different discretizations of the reward function. M5 does not need to discretize by hand the reward space and it obtains better or similar results than J48. We have shown that 2SRL can be used in complex domains with continuous rewards obtaining better results than using only the first phase of the algorithm.

Acknowledgements

This work has been partially sponsored by a regional project CCG08-UC3M/TIC-4141 of the *Comunidad de Madrid*, and a national project TIN2008-06701-C03-03 of the *Ministerio de Ciencia e Innovación* of Spain.

References

1. Bellman, R.: Dynamic programming. Princeton University Press, Princeton (1957)
2. Boyan, J.A., Moore, A.W.: Generalization in reinforcement learning: Safely approximating the value function. Adva. Neural Inform. Process. Syst. 7 (1995)
3. Fernández, F., Borrajo, D.: VQQL. Applying vector quantization to reinforcement learning. In: Veloso, M.M., Pagello, E., Kitano, H. (eds.) RoboCup 1999. LNCS, vol. 1856, pp. 292–303. Springer, Heidelberg (2000)
4. Frank, E., Witten, I.H.: Generating Accurate Rule Sets Without Global Optimization. In: Fifteenth International Conference on Machine Learning, pp. 144–151 (1998)
5. Fernández, F., Borrajo, D.: Two steps reinforcement learning. Internation Journal of Intelligent Systems 23, 213–245 (2008)
6. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement Learning: A Survay (1996)
7. Smart, W.D.: Making reinforcement learning work on real robots. PhD thesis, Department of Computer Science at Brown University, Providence, RI (May 2002)
8. Stone, P., Sutton, R., Kuhlmann, G.: Reinforcement Learning for RoboCup-Soccer Keepaway. Adaptive Behaviour 13(3) (2005)
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
10. Frank, E., Witten, I.H.: Data Mining: Practical Machine Learning Tools and Techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
11. Munos, R., Moore, A.: Variable Resolution Discretization in Optimal Control. In: Machine Learning, pp. 291–323. Kluwer Academic Publishers, Dordrecht (2002)