

Q -Learning with Double Progressive Widening : Application to Robotics

Nataliya Sokolovska, Olivier Teytaud, and Mario Milone

INRIA Saclay, CNRS UMR 8623 & LRI, Université Paris Sud, Orsay, France

Abstract. Discretization of state and action spaces is a critical issue in Q -Learning. In our contribution, we propose a real-time adaptation of the discretization by the progressive widening technique which has been already used in bandit-based methods. Results are consistently converging to the optimum of the problem, without changing the parametrization for each new problem.

Key words: Q -Learning, discretization, applications

1 Introduction

In a large number of real world applications it is intractable to estimate a model. Q -Learning is a well known model-free reinforcement learning algorithm, where Q -values – which estimate the expected reward for taking a particular action in a given state – are learnt. However, in the approach it is assumed that the domain is discrete, or discretized. If the state and/or action spaces are continuous, the application of the Q -Learning is not straightforward. If the state/action domains are continuous (or very large), it becomes hardly possible to keep (and to update) a look-up table which contains Q -values for each state-action pair. Besides discretization approaches [1,2,3], including adaptive techniques, there exist a number of techniques applied to the reinforcement learning which allow to work with continuous values. To discretize a continuous state and action space is a challenge, since if a discretization is too rough, it will be impossible to find the optimal policy; if a grid is too fine, the generalization will be lost.

Among the state-of-the art approaches are the following discretizing (and often feature and model selecting) approaches. A historical but still actively exploited approach is CMAC (Cerebellar Model Arithmetic Computer) that has been introduced for robotics [4,5]. In CMAC Q -Learning the state space is partitioned into tiles, which are binary features. A parameter (or weight) θ is associated with each tile, and Q -values are not kept in a look-up table but are represented by a parametric family of functions, parametrized by the vector Θ . CMAC Q -Learning is similar to a neural network. Overall, the introduction of neural networks into Q -Learning to process continuous states has been actively studied, see e.g., [6]. If the number of tiles is quite large, the computational complexity of such parametric approaches can be high.

Vector Quantization Q -Learning (VQQL) [7,8] produces a compact representation of the state domain via clustering of simulated states. The drawback of the method is that a refinement of the grid is not foreseen.

Recently in [9] the Two Steps Reinforcement Learning (2SRL) has been introduced, where decision (and regression) trees are used to perform the state space discretization. The algorithm is based on two alternating steps; in the first phase some discretization of the state domain is produced, in the second one a current policy is improved. The reported drawback of the method is the requirement for the discrete reward function.

The important problem is to refine the discretization grid of states (and actions) adaptively, especially around the areas of interest, e.g., around the goal. If the initial grid is rather coarse, and if all vertices of the grid are far enough from the goal, it is possible that an agent never reaches a goal. An adaptive approach to refine the initial grid has been recently proposed by [10]. The idea is to provide pseudo-goals which lie on the vertices of the initial grid. It has been shown that the method is efficient, however, its serious disadvantage is that the knowledge of a location of a goal is required. The initialization of the grid with the pseudo-goals, which are in a proximity to the true goal, is not obvious.

In this contribution, we propose a technique, inspired by methodologies developed in Monte-Carlo Tree Search, for directly working in the continuous setting. To the best of our knowledge, this is the first dynamic discretization approach handling both continuous states and continuous actions - which is critical for many important applications.

The paper is organized as follows. Section 2 presents progressive widening Q -Learning. Section 3 provides the results of our experiments on both a synthetic reinforcement learning task and on a realistic problem of a robot navigation in a 3-dimensional partially observable environment. Concluding remarks and perspectives close the paper.

2 Progressive Widening Q -Learning

In this section, we describe briefly the Q -Learning approach. We discuss its limits with respect to continuous problems, and introduce progressive widening Q -Learning procedure.

2.1 Q -Learning Approach

To solve a goal-planning task means to find an optimal policy, i.e. a policy π^* that is equal or better (in terms of cumulated expected reward) than any other policy π . It is known [11] that optimal policies share the same optimal action-value function Q^* . Given a set of states \mathcal{S} and a set of actions \mathcal{A} , optimal

action-value function is defined as

$$\begin{aligned}
 Q^*(s, a) &= \max_{\pi} Q^{\pi}(s, a) \\
 &= \mathbb{E}\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t = s, a_t = a\} \\
 &= \sum_{s'} \mathcal{P}_{ss'}^a \left(\mathcal{R}_{ss'}^a + \gamma \max_{a'} Q^*(s', a') \right), \tag{1}
 \end{aligned}$$

where $\mathcal{P}_{ss'}^a = p(s'|s, a)$, \mathcal{R} is the reward, $s, s' \in \mathcal{S}$, and $a, a' \in \mathcal{A}$. In other words, the action-value function $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ defines the quality of each (state, action) pair.

Q-Learning is a general term for approaches which compute the expected reward given an action a in a given state s , and allow to choose an action maximizing the reward value. The strength of Q-Learning methods is that they do not require any knowledge of a model of environment $\mathcal{P}_{ss'}^a$, which is not available in a number of real-world applications.

An example of a policy is the greedy policy, given by

$$\pi(s) = \arg \max_a Q^{\pi}(s, a), \tag{2}$$

which we use in the following.

One-step Q-Learning is proposed by [12]. The approach is based on the following update rule:

$$Q(s, a) = Q(s, a) + \alpha \left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right), \tag{3}$$

where α is usually called learning rate, $\alpha \in]0; 1]$, and γ – discount factor, $\gamma \in [0; 1[$. The complete reinforcement learning procedure is drafted as Algorithm 1.

Algorithm 1 One-step Q-Learning

```

Initialize  $Q(s, a)$ 
for each episode do
  Initialize  $s$ 
  for each step of episode do
    Choose  $a$  ( $\epsilon$ -greedy policy derived from  $Q$ )
    Take action  $a$ , observe  $r$  and  $s'$ 
     $Q(s, a) = Q(s, a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)$ 
     $s = s'$ 
  end for
end for

```

2.2 Applying Double Progressive Widening to Q-Learning

Since we can not enumerate all possible states and all possible actions, we exploit the idea to use a table of Q values which is not static. We explore and add states

and actions to the Q -table progressively. Under double progressive widening we mean that we increase the number of explored states and actions.

We can naturally apply the double progressive widening procedure to the Q -Learning framework. Since our state and action spaces both are continuous, some discretization has to be done, if we want to apply the Q -Learning directly. At the same time, both the state and action spaces should be well explored to achieve some reasonable cumulated reward. For this, we slowly increase the number of states and actions, by the progressive widening technique successfully used in bandit-based algorithms [13,14,15]. When a state is sufficiently highly visited, compared to the number of times the previous action has been tried in the previous state, then it is added in the discretization of states; and when the number of visits of a state is sufficiently large, compared to the pool of actions already considered, then a new action is added. The discretization of states and actions is carried out based on the Euclidean distance. A newly observed state (action) gets the same discrete value as its closest state (action) in the already explored set of states \mathcal{S} (set of actions \mathcal{A}).

The approach we use is drafted as Algorithm 2; λ is the progressive widening parameter associated with states exploration, and λ' – with exploration of decisions. Both parameters are equal in our experiments, so that we do not introduce a bias by a highly tuned parameterization; interestingly, we will see that some values of $\lambda = \lambda'$ are good for all our tests.

Algorithm 2 Progressive Widening Q -Learning

```

Initialize  $\mathcal{S}$  – set of explored states
Initialize  $\mathcal{A}$  – set of explored actions, specific for states
Initialize  $Q(s, a)$  –  $Q$ -values
Initialize parameters  $C > 0$ ,  $\lambda \in ]0, 1[$ ,  $\alpha \in ]0, 1[$ ,  $\gamma \in [0; 1[$ 
for each episode do
  Initialize  $s$ 
  for each step of episode do
     $nbVisits(s) = nbVisits(s) + 1$ 
     $k = \lceil CnbVisits(s)^\lambda \rceil$ 
    Choose action  $a$  from  $\{a_1, \dots, a_k\}$  associated with  $s$  using Eq. (2)
    Update the number of visits  $nbVisits(s, a) = nbVisits(s, a) + 1$ 
    Take action  $a$ , observe  $s'$  and  $r$ 
    if  $(\lceil CnbVisits(s, a)^{\lambda'} \rceil > \#\mathcal{S}) \& (s' \notin \mathcal{S})$  then
       $\mathcal{S} = \mathcal{S} \cup s'$ 
    end if
    Update  $Q(s, a)$ , using Eq. (3)
     $s = \arg \min_{s'' \in \mathcal{S}} \|s'' - s'\|$ 
  end for
end for

```

3 Experiments

In this section, we illustrate efficiency of the proposed progressive widening Q -Learning on a synthetic problem called Treasure Hunt, and on a realistic – robot navigation – task.

3.1 Treasure Hunt Problem

The problem considered in this section is an artificial problem which allows us to demonstrate clearly that the double progressive widening Q -Learning is a powerful approach. Treasure Hunt is a two-dimensional problem, i.e., an agent moves in a two-dimensional environment, with the dimensions 15×15 . The aim of the agent is to discover the treasure. Both the agent and the treasure are always initialized in the same coordinates (the agent is initialized in the lower left corner of the 15×15 room, and the treasure is located in the upper right corner). The agent knows his position at each time step. The reward equals 1000 when the treasure is reached, otherwise the instantaneous reward equals -1 .

To consider different scenarios, we make use of three variations of the Treasure Hunt framework:

1. Treasure Hunt as described above
2. An obstacle is added, i.e. a hole is added in the center of the two-dimensional space. If the agent falls into the hole, the reward is -500 .
3. Uniform noise (on states) is added

The difficulty is the fact that both states (positions of the agent) and actions (or decisions of the agent) are continuous. In our experiments, we compare the standard Q -Learning without progressive widening with the method proposed above. To perform experiments with the standard Q -Learning we discretize the state and action spaces using some grid of a constant size. The progressive widening parameter λ controls how many and how fast new states and actions are added to the table, which contains states \times actions Q -values. In our experiments, we apply the same progressive widening parameter value to states and actions, i.e., $\lambda = \lambda'$. Note, that when $\lambda = 0$, neither new states, nor new actions are added, and therefore, the results are equivalent to ones of the standard Q -Learning. We can start the progressive widening procedure from empty sets for states and actions. In the experiments, we have considered two cases: we start the learning procedures (Q -Learning and progressive widening Q -Learning) from 1 pre-simulated state and action; and from 5 pre-simulated states and actions. We use the following Q -Learning parameters in our experiments: $\alpha = 0.15$, $\gamma = 0.85$.

The following figures illustrate our results – the mean of the cumulated reward – on different scenarios of the Treasure Hunt problem and for different progressive widening (PW in the legend) parameter values. The number of Monte-Carlo simulations on the plots is 500. Figure 1 is the basic case, where there are no any obstacles and the transitions are completely deterministic (no noise). Figure 2 demonstrates the case with the noise. Figure 3 is the variation

with the hole in the center of the 2-dimensional search space, and Figure 4 is the scenario with the obstacle and with the noise. We can conclude that the standard Q -Learning with 1 pre-simulated state and action copes quite bad with the task. The Q -Learning which disposes 5 states and 5 actions ameliorates the performance when the number of the Learning episodes increases. It is obvious that the double progressive widening Q -Learning is much more efficient. Note, that the progressive widening parameter plays a significant role, $\lambda = 0.5$ allows to add more states and actions than $\lambda = 0.25$, and hence makes the discretization of the state and action spaces more adapted. Overall, the problem is rather simple and can be discretized in a small number of states and actions.

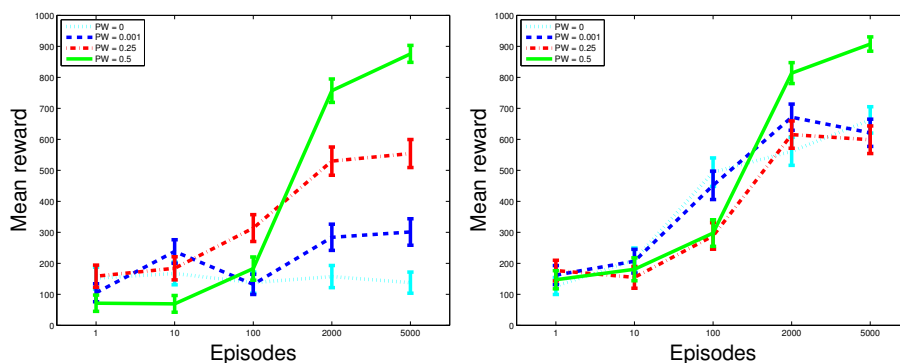


Fig. 1. Treasure Hunt task. Without noise and without obstacles; left: starting with 1 state and 1 action; right: starting with 5 states and 5 actions. Note that while applying the parameter 0, the double progressive widening boils down to a non-adaptive discretization.

3.2 Robot Navigation in a 3D Partially Observable Environment

Autonomous robot navigation is a challenging task, especially in an environment which is partially observable. The values of the Q -Learning parameters and the number of simulations are the same as in the previous section.

The 3D simulator we use in our experiments has been developed at IDIAP¹. The simulator models the 3D environment which resembles one designed for computer video games. An agent is placed in a virtual room, and the goal is to teach it to touch the red flag. In every training (and testing) episode, the robot and the flag are placed randomly. The described task is a typical problem which can be solved by reinforcement learning. If the robot touches the walls, the instantaneous reward equals -1 , if the red flag is reached the reward is $+10$, otherwise the reward is 0 at each time step.

¹ <http://www.idiap.ch/>

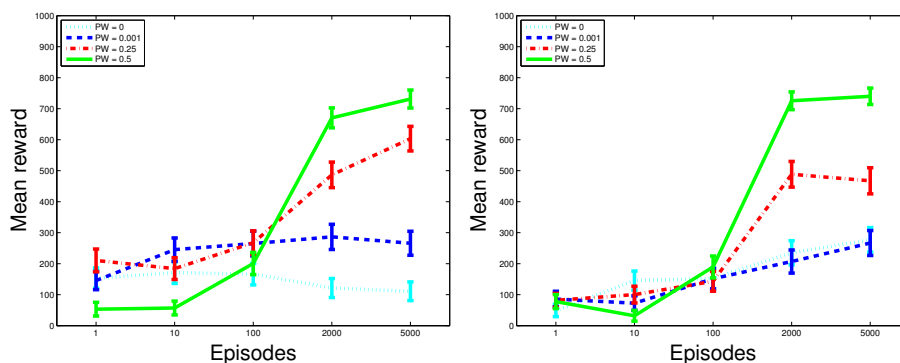


Fig. 2. Treasure Hunt task. With noise and without obstacles; left: starting with 1 state and 1 action; right: starting with 5 states and 5 actions. Note that while applying the parameter 0, the double progressive widening boils down to a non-adaptive discretization.

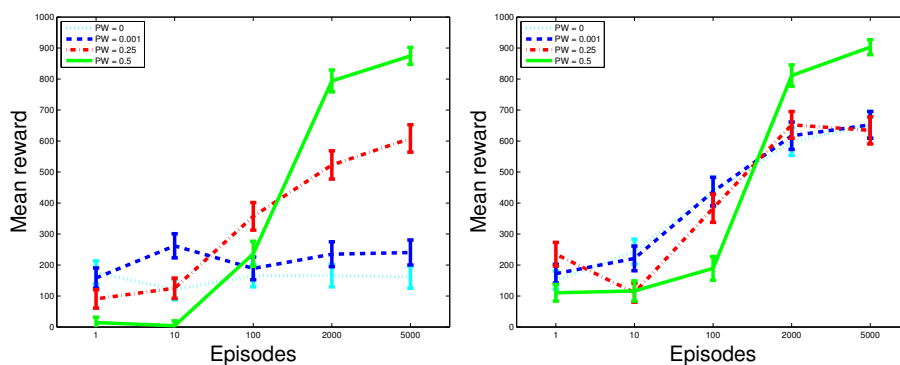


Fig. 3. Treasure Hunt task. Without noise and with obstacles; left: starting with 1 state and 1 action; right: starting with 5 states and 5 actions. Note that while applying the parameter 0, the double progressive widening boils down to a non-adaptive discretization.

The difficulty of the task is that the environment is not fully observable, it is partially observable. The robot does not know its position. The agent has to deduce where it is and what it has to do (i.e., which action should be taken) based on video images it gets on each time step. Figure 5 illustrates two typical observations of an agent (on the left: the robot does not observe the goal, on the right: the robot sees the flag).

On each time step, the agent gets a video image. Using the a priori knowledge that the flag, the robot is looking for, is red, we apply the image processing technique to extract the information, whether the agent sees the flag, whether the flag is observed on the right/left/in front of the robot. We have introduced

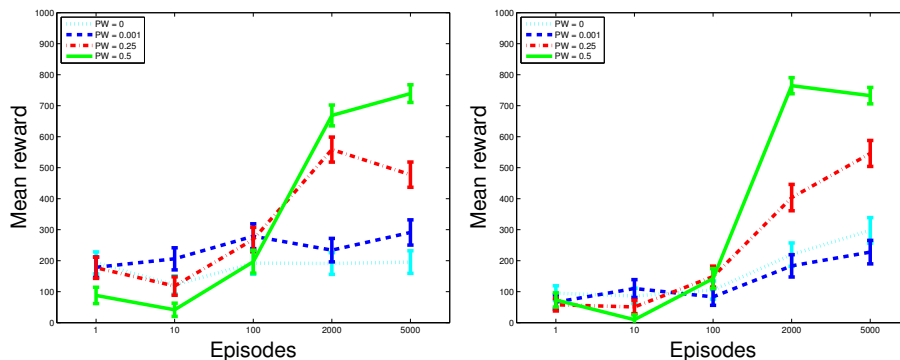


Fig. 4. Treasure Hunt task. With noise and with obstacles; left: starting with 1 state and 1 action; right: starting with 5 states and 5 actions. Note that while applying the parameter 0, the double progressive widening boils down to a non-adaptive discretization.



Fig. 5. 3D environment designed by the simulator. On the left: the robot does not observe the goal, on the right: the robot sees the flag.

an additional reward for the training phase only. If the flag is observed on the right or on the left, the supplementary reward is +2.5, if the robot can observe the goal just in front of it, the supplementary reward equals +5. Figure 6 shows the dependence of the cumulated reward (test phase) on the learning time.

In the Appendix we provide video images of a typical exploitation trajectory of the agent. The robot and the goal are initialized and placed randomly, and the agent is approaching the the red flag.

4 Conclusion

We proposed, to the best of our knowledge, the first approach for handling, without fixed discretization, both a continuous state space and a continuous action space. While we have no consistency proof, we believe that results as

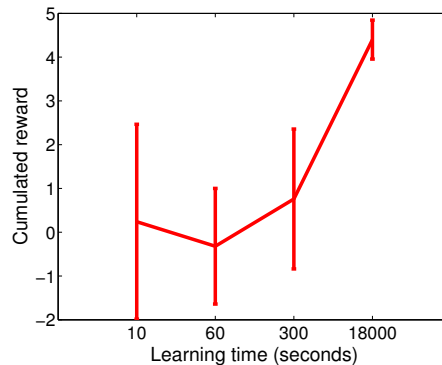


Fig. 6. Navigation of the robot in a 3D environment: cumulated reward (testing phase); $\lambda = 0.5$

in [16] can be used for proving the consistency of our approach. Experimental results suggest that the algorithm efficiently adapts the discretization where it is needed, and that the “widening” principle, consisting in extending an edge when it is simulated more than n^λ times where n is the number of visits to the parent situation, is a stable methodology with coefficient λ around $\frac{1}{3}$ (interestingly, nearly the same constant as in [17] and [16] in different contexts).

Acknowledgements

We are grateful to the FP7 program (European project MASH – Massive Sets of Heuristics) for support.

References

1. Davies, S.: Multidimensional Triangulation and Interpolation for Reinforcement Learning. In: *Advances in Neural Information Processing Systems* (1997)
2. Munos, R., Moore, A.: Variable Resolution Discretization in Optimal Control. Technical report, Robotics Institute, CMU (1999)
3. Munos, R., Moore, A.W.: Variable Resolution Discretization for High-accuracy Solutions of Optimal Control Problems. In: *IJCAI*, pp. 1348–1355 (1999)
4. Albus, J.S.: A New Approach to Manipulator Control: the Cerebellar Model Articulation Controller. *Journal of Dynamic Systems, Measurement, and Control* 97, 220–227 (1975)
5. Burgin, G.: Using Cerebellar Arithmetic Computers. In: *AI Expert* 7 (1992)
6. Gaskett, C., Wettergreen, D., Zelinsky, A.: *Q-learning in Continuous State and Action Spaces*. In: *Australian Joint Conference on Artificial Intelligence*, Springer-Verlag, pp. 417–428 (1999)
7. Gersho, A., Gray, R.M.: *Vector Quantization and Signal Compression*. Kluwer Academic Publishers (1991)
8. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement Learning for Robocup-soccer Keepaway. *Adaptive Behavior* 3, 165–188 (2005)

9. Fernández, F., Borrajo, D.: Two Steps Reinforcement Learning. *International Journal of Intelligent Systems* 2, 213–245 (2008)
10. Lampton, A., Valasek, J.: Multiresolution State-Space Discretization Method for *Q*-Learning. In: *American Control Conference* (2009)
11. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: an Introduction*. MIT Press (1998)
12. Watkins, C.J.C.H.: *Learning from Delayed Rewards*. PhD thesis, Cambridge University (1989)
13. Couëtoux, A., Hooek, J.B., Sokolovska, N., Teytaud, O., Bonnard, N.: Continuous Upper Confidence Trees. In: *International Conference on Learning and Intelligent Optimization* (2011)
14. Coulom, R.: Monte-Carlo Tree Search in Crazy Stone. In: *Game Programming Workshop* (2007)
15. Rolet, P., Sebag, M., Teytaud, O.: Boosting Active Learning to Optimality: a Tractable Monte-Carlo, Billiard-based Algorithm. In: *European Conference on Machine Learning* (2009)
16. Wang, Y., Audibert, J.Y., Munos, R.: Algorithms for Infinitely Many-armed Bandits. In: *Advances in Neural Information Processing Systems* (2008)
17. Coulom, R.: Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In P. Ciancarini and H. J. van den Herik, editors, *Proceedings of the 5th International Conference on Computers and Games, Turin, Italy* (2006)

5 Appendix: Exploitation Trajectory of the Agent

