Neurocomputing 74 (2011) 1251–1259

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Robust high performance reinforcement learning through weighted k-nearest neighbors

José Antonio Martín H [a,*], Javier de Lope [b], Darío Maravall [c]

[a] Depto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Spain
[b] Dept. Applied Intelligent Systems, Universidad Politécnica de Madrid, Spain
[c] Dept. Artificial Intelligence, Universidad Politécnica de Madrid, Spain

ARTICLE INFO

Available online 20 October 2010

Keywords:
Reinforcement learning
k-Nearest-neighbors
Function approximation
Temporal-difference learning

ABSTRACT

The aim of this paper is to present (jointly) a series of robust high performance (award winning) implementations of reinforcement learning algorithms based on temporal-difference learning and weighted k- nearest neighbors for linear function approximation. These algorithms, named kNN-TD(λ) methods, where rigorously tested at the Second and Third Annual Reinforcement Learning Competitions (RLC2008 and RCL2009) held in Helsinki and Montreal respectively, where the kNN-TD(λ) method (JAMH team) won in the PolyAthlon 2008 domain, obtained the second place in 2009 and also the second place in the Mountain-Car 2008 domain showing that it is one of the state of the art general purpose reinforcement learning implementations. These algorithms are able to learn quickly, to generalize properly over continuous state spaces and also to be robust to a high degree of environmental noise. Furthermore, we describe a derivation of kNN-TD(λ) algorithm for problems where the use of continuous actions have clear advantages over the use of fine grained discrete actions: the Ex$\langle a \rangle$ reinforcement learning algorithm.

1. Introduction

Reinforcement learning (RL) [11,16] is a paradigm of machine learning (ML) in which rewards and punishments guide the learning process. In RL there is an Agent (learner) which acts autonomously and receives a scalar reward signal that is used to evaluate the consequences of its actions. The framework of RL is designed to guide the agent in maximizing the expected future cumulative (or average) reward received from the environment.

Eq. (1) is the classical temporal-difference (TD) learning rule:

$$Q(s,a)_{t+1} = Q(s,a)_t + \alpha[x_t - Q(s,a)_t], \tag{1}$$

where

$$x_t = r_{t+1} + \gamma \max_a Q(s_{t+1},a)_t \quad [19,\ Q\text{–Learning}], \tag{2}$$

$$x_t = r_{t+1} + \gamma Q(s_{t+1},a_{t+1})_t \quad [16,\ \text{SARSA}]. \tag{3}$$

This rule (1) comprises three strong ideas which are central to RL: (i) the notion that knowledge is, in essence, made of expectations (knowledge is prediction), (ii) the temporal-difference notion [17] which is a general method of prediction very well suited to incremental learning from immediate experience (the steps in a sequence should be evaluated and adjusted according to their immediate or near immediate successors, rather than according to the final outcome [17]) and (iii) the reward hypothesis (a.k.a. the reinforcement learning hypothesis) that states that "all of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal called reward" [18].

In this paper we present a family of reinforcement learning algorithms. These algorithms are developed using the classical formulation of TD-learning and a k-nearest neighbors scheme for the perception, action selection, expectations memory and learning processes.

The perceptual process of the proposed methods is defined within a kind of k-nearest neighbors approach in order to produce a probabilistic representation of the input signal to construct robust state descriptions based on a collection (knn) of receptive field units and a probability distribution vector $p(knn)$ over the knn collection. The action selection mechanisms of the proposed methods can be understood, in k-NN terminology, as that the Agent will select the most "frequent" recommended action in the neighborhood of the observed state s or, in Bayesian terms, the action for which there is more evidence of a high future reward. Finally, the learning mechanism can be understood as a kind of temporal difference back-propagation which assigns the right TD error to each neighbor

* Corresponding author. Tel.: +34 91 394 7600; fax: +34 91 394 7510.
E-mail addresses: jamartinh@fdi.ucm.es (J.A. Martín H), javier.delope@upm.es (J. de Lope), dmaravall@fi.upm.es (D. Maravall).

0925-2312/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2010.07.027

according to its respective influence in the decision making and action selection procedures, which is represented by its probability of being the nearest neighbor of the observed states.

The k-nearest neighbors (k-NN) technique [6–8] is a method for classifying objects based on closest training examples in the feature space. The training examples are mapped into a multidimensional feature space. The space is partitioned into regions by class labels of the training samples. A point in the space is assigned to a class if it is the most frequent class label among the k-nearest samples used in the training phase. The determinant parameters in k-NN techniques are the number k which determines how many units are to be considered as the neighbors and the distance function, generally the Euclidian distance is used.

It has been proved [7] that there is a close relation between nearest neighbors methods and the Bayesian theory of optimal decision making. One of the most relevant results on this line is the Bayesian error bound for the nearest neighbors method which establishes that the upper bound on the probability of error for the NN-rule is less than twice the Bayes' error rate [7].

There has been other uses of the k-NN rule, e.g. in dynamic programming (DP) [9] for function approximation with successful results. Also, we must mention that there is a relation between the kind of algorithms that we propose and the so-called "locally weighted learning" (LWL) methods surveyed by Atkeson and Moore [4]. LWL approximations can also be viewed as a particular kind of artificial neural network (ANN) for approximating smooth functions [5].

In this paper, we show the use of the k-NN technique in the development of three RL algorithms: the kNN-TD, kNN-TD(λ) and the Ex$\langle a \rangle$ algorithm. Also we present an online adaptive filter, the $k-$NNδs online adaptive filter, for dealing with highly noisy environments. These algorithms are based on some preliminary works [12–14]. Public (open-source) implementations of these algorithms are published in the web page http://www.dia.fi.upm.es/~jamartin/download.htm in order that these tools be used, evaluated or improved by the interested researchers.

The rest of the paper is organized as follows: Section 2 presents a basic and very simple TD-learning algorithm based on the simple Nearest Neighbor approach and describes some issues related to this algorithm. Section 3 presents the details of the kNN-TD and the kNN-TD(λ) algorithms. In Section 4 the continuous actions Ex$\langle a \rangle$ Reinforcement Learning algorithm is presented. The experimental results and performance remarks are presented in Section 6. Finally in Section 7 the conclusions and further work are presented.

2. A basic TD-learning algorithm by using simple nearest neighbor

By using the simple nearest neighbors approach a very basic but general enough TD-learning algorithm could be developed.

The first task to address is the definition of the points which will act as the neighbors of the observations (s). Although there are many ways of covering the space, a simple procedure is to generate points uniformly distributed over the complete state space.

Finally for each defined point x_i in the state space will be an associated action-value predictor $Q(i,a)$.

A pseudo-code of the NN-TD RL algorithm is presented in Algorithm 1.

Algorithm 1. A simple nearest neighbors TD algorithm.

```
1:     Initialize the space of points $X \in R^n$
2:     Initialize Q arbitrarily
3:     **repeat** {for each episode}
4:        Initialize $s$
5:        $x \leftarrow$ nearest neighbor of $s$
6:        $i \leftarrow indexof(x)$
7:        Chose $a$ from $s$ according to $Q(i,a)$
8:        **repeat** {for each step of episode}
9:           Take action $a$, observe $r$, $s'$
10:          $x' \leftarrow$ nearest neighbor of $s'$
11:          $j \leftarrow indexof(x')$
12:          Chose $a'$ from $s'$ according to $Q(j,a')$
13:          $Q(i,a) \leftarrow Q(i,a) + \alpha[r + \gamma Q(j,a') - Q(i,a)]$
14:          $a \leftarrow a'$, $i \leftarrow j$
15:       **until** $s$ is terminal
16:    **until** learning ends
```

However, we can see soon some circumstances for which this simple approach will not succeed:

1. When using a very fine grained state-space, since the learning may become slow when not intractable at all.
2. When behaving on a noisy environment, the algorithm may not converge properly due to the impossibility to select the right neighbors to calculate the value function or to update the right ones.
3. When behaving on a nondeterministic transitions environment, due to, basically the same reasons as in (2).

In this work we propose the use of a weighted k-nearest neighbors approach to overcome these issues.

3. The *k* NN-TD reinforcement learning algorithm

The algorithm description will follow the classical interaction cycle in RL. The basic interaction cycle in RL could be described by four basic steps as shown in Fig. 1:

1. Perceive the state s of the environment.
2. Emit an action a to the environment based on s.
3. Observe the new state s' of the environment and the reward r received.
4. Learn from the experience based on s, a, s' and r.

3.1. Perception

The first task is to determine the k-nearest neighbors set (knn) of the current observation s.

It is common that the scale in some dimensions do not match leading to a biased selection of the k-nearest neighbors set over the dimension whose magnitude is smaller. For avoiding this potential bias it is necessary the correct standardization of all dimensions in such a way that its observed values be in the same scale. This standardization could be achieved by mapping all the points

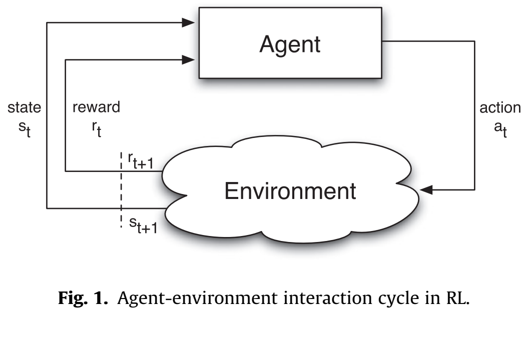

Fig. 1. Agent-environment interaction cycle in RL.

over a same range space, for instance, by applying the rule (4) which maps all points from the state space X to a space $\dot{x}$ in the range $[-1,1]$:

$$\dot{x} = 2\left(\frac{x-\min(X)}{\max(X)-\min(X)}\right)-1. \tag{4}$$

Also the observations s should be then mapped into the same space when determining the k-nearest neighbors set.

As in the simple NN method, for each defined neighbor will be an associated action-value predictor $Q(i,a)$ and also an activation coefficient w_i which is calculated as follows:

$$w_i = \frac{1}{1+d_i^2} \quad \forall i \in [1,\ldots,k], \tag{5}$$

where d_i^2 is a distance function and then for each time step will be k active neighbors whose weights are inverse to the distance from the current observation (s).

The second task is to obtain the probability distribution $p(knn)$ over knn. We must note that each element in the knn set will provide information on the current observed state s. This information could be in the form of an associated value (i.e. an associated action-value predictor $Q(i,a)$) or indeed implicitly defined just by the meaning of being x_i a particular neighbor of s. In any respect, the probabilities are implied by the current weights vector (w). This vector must be normalized in order to express a probability distribution $p(knn)$, thus the probabilities for each element in the knn set are calculated by

$$p(i) = \frac{w_i}{\sum w_i} \quad \forall i \in knn. \tag{6}$$

Then, a perceptual representation, or simply a perception, i.e. an internal representation of a particular situation of the external world (environment), is uniquely and completely defined by means of just two components (knn,p) obtained from a current observation (s):

1. A set knn which contains the k-nearest neighbors of the observation s, which can be seen as a collection of activated feature detector units.
2. A probability distribution vector p of each element in the knn set assigning a particular proportional activation to each feature detector unit.

3.2. Action selection

Once obtained a perceptual representation (knn,p) of an observed state (s), the agent should emit an action (a) to the environment. For doing so, the agent has to apply some "rational" action selection mechanism based on its expectation of future cumulative reward for taking action (a) in s. In RL the problem of balancing the relation of exploration/exploitation is achieved explicitly by using the so-called action–selection mechanisms with corresponding methods like greedy, $\varepsilon-$greedy, soft-max methods, etc., and can be found in the standard literature [16]. We will focus on computing the reward expectations (i.e. predicting future reward) for each action by means of a collective prediction process which can then be used directly by any classical action selection mechanism.

The objective of the prediction process consists of calculating the expected value of the predictions for each available action a_i in the observed state s. Since this process involves many different predictions, one for each element in the knn each one with its respective weight $p(i)$, the procedure gets reduced to estimate an expected value, that is:

$$\mu = \sum_{i=1}^{n} x_i p(x_i). \tag{7}$$

In this way, the expected value of the learning target for a given action a is estimated by

$$\langle \mathcal{Q}(knn,a)\rangle = \sum_{i=1}^{knn} Q(i,a)p(i), \tag{8}$$

hence we can see clearly that $p(i)$ acquires the meaning of the probability $P(\mathcal{Q}(knn,a)=Q(i,a)|s)$ that the $\mathcal{Q}(knn,a)$ takes the value $Q(i,a)$ given the observation (s) of the environment. Then the action selection mechanism can directly derive a greedy policy from the expectations for each perceptual representation as

$$a_{greedy} = \operatorname*{argmax}_a \mathcal{Q}(knn,a). \tag{9}$$

3.3. Learning

The learning process involves the use of the accumulated experience and thus it will rely on past and current experienced observations, actions performed and received reward (s, a, s' and r). Given that the Agent have experienced the state s and performed an action a and then observed a new state s' and received a reward r there are two perceptual representations, one for each state respectively. These perceptual representations are the tuples (knn,w) for observation s and (knn',w') for observation s'.

Following the TD-learning scheme we need an estimation of the future cumulative reward. This estimation is generally obtained by the action-value of the current state s', i.e. $\max_a \mathcal{Q}(knn',a)$ in off-policy learning methods and by $\mathcal{Q}(knn',a')$ in on-policy learning.

For whichever (off/on policy) followed method, an expected (predicted) value is calculated:

$$\langle \mathcal{Q}(knn',a)\rangle = \sum_{i=1}^{knn'} Q(i,a)p'(i), \tag{10}$$

where knn' is the set of the k-nearest neighbors of s', $p'(i)$ are the probabilities of each neighbor in the set knn'. In this way, the TD-error (δ) can be calculated as (on/off policy respectively)

$$\delta = r+\gamma\mathcal{Q}(knn',a')-\mathcal{Q}(knn,a), \tag{11}$$

$$\delta = r+\gamma\max_a\mathcal{Q}(knn',a)-\mathcal{Q}(knn,a). \tag{12}$$

Thus we can simply update the prediction of each point in the knn set by applying the update rule

$$Q(i,a)_{t+1} = Q(i,a)_t+\alpha\delta p(i) \quad \forall i \in knn, \tag{13}$$

note that the probability term $p(i)$ is just the gradient term used in temporal difference methods for the linear function approximation case (see [16] for details of the linear function approximation case).

A pseudo-code of the kNN-TD RL algorithm is presented in Algorithm 2.

Algorithm 2. The kNN-TD reinforcement learning control algorithm.

```
1:   Initialize the space of points $X \in R^n$
2:   Initialize Q arbitrarily
3:   **repeat** {for each episode}
4:     Initialize $s$
5:     $knn \leftarrow k-$nearest neighbors of $s$
6:     $p(knn) \leftarrow$ probabilities of each $i \in knn$
7:     $\mathcal{Q}(knn,a) = Q(knn,a) \cdot p(knn)$
8:     Chose $a$ from $s$ according to $\mathcal{Q}(knn,a)$
9:     **repeat** {for each step of episode}
10:      Take action $a$, observe $r$, $s'$
```

```
11:         $knn' \leftarrow$ $k$-nearest neighbors of $s'$
12:         $p(knn') \leftarrow$ probabilities of each $i \in knn'$
13:         Chose $a'$ from $s'$ according to $\mathcal{Q}(knn',a)$
14:         $\mathcal{Q}(knn',a') = Q(knn',a') \cdot p(knn')$
15:         $\delta \leftarrow r + \gamma \mathcal{Q}(knn',a') - \mathcal{Q}(knn,a)$
16:         for all $i \in knn$ do
17:             $Q(i,a) \leftarrow Q(i,a) + \alpha\delta p(i)$
18:         end for
19:         $a \leftarrow a'$, $s \leftarrow s'$, $knn \leftarrow knn'$
20:     until $s$ is terminal
21: until learning ends
```

3.4. Probability traces $e(i) \leftarrow p(i)$: kNN-TD(λ)

The use of eligibility traces [16, p. 161] is a useful technique for improving the performance of RL algorithms. Eligibility traces are controlled by a parameter λ that takes values in the range [0,1]. This parameter makes that the methods based on eligibility traces behave between two extremes: one-step TD-learning ($\lambda=0$) and Monte Carlo methods ($\lambda=1$) [16, p. 163]. A pseudo-code of the kNN-TD(λ) RL algorithm is presented in Algorithm 3.

Algorithm 3. The kNN-TD(λ) reinforcement learning control algorithm.

```
1:  Initialize the space of points $X \in R^n$
2:  Initialize Q arbitrarily and $e \leftarrow 0$
3:  repeat {for each episode}
4:      Initialize $s$, $e \leftarrow 0$
5:      $knn \leftarrow$ $k$-nearest neighbors of $s$
6:      $p(knn) \leftarrow$ probabilities of each $cl \in knn$
7:      $\mathcal{Q}(knn,a) = Q(knn,a) \cdot p(knn)$
8:      Chose $a$ from $s$ according to $\mathcal{Q}(knn,a)$
9:      repeat {for each step of episode}
10:         Take action $a$, observe $r$, $s'$
11:         $knn' \leftarrow$ $k$-nearest neighbors of $s'$
12:         $p(knn') \leftarrow$ probabilities of each $i \in knn'$
13:         Chose $a'$ from $s'$ according to $\mathcal{Q}(knn',a)$
14:         $\mathcal{Q}(knn',a') = Q(knn',a') \cdot p(knn')$
15:         Update Q and e:
16:             $e(knn,\cdots) \leftarrow 0$ {optional}
17:             $e(knn,a) \leftarrow p(knn)$
18:             $\delta \leftarrow r + \gamma \mathcal{Q}(knn',a') - \mathcal{Q}(knn,a)$
19:             $Q \leftarrow Q + \alpha\delta e$
20:             $e \leftarrow \gamma\lambda e$
21:         $a \leftarrow a'$, $s \leftarrow s'$, $knn \leftarrow knn'$
22:     until $s$ is terminal
23: until learning ends
```

In order to implement eligibility traces it is necessary to define a matrix e in which the traces for each neighbor will be stored. A consequence of the use of eligibility traces is that the update rule should be applied to all the state-space and not only to the knn set. Thus the update rule for the case of eligibility traces becomes

$$Q_{t+1} = Q_t + \alpha\delta e. \tag{14}$$

At least two kinds of traces can be defined: cumulating traces and replacing traces.

Cumulating traces are defined in a simple form

$$e(knn,a)_{t+1} = e(knn,a)_t + p(knn), \tag{15}$$

where knn is the set of k-nearest neighbors of the observation s and $p(knn)$ are its corresponding probabilities.

In general, replacing traces are more stable and perform better than cumulating traces [15,16]. Replacing traces in the kNN-TD(λ) algorithm are defined in a simple way by establishing the value of the trace for each neighbor as its probability $p(i)$ for the performed action (a) while for actions different from a the traces reset to 0. This form of trace resetting is a specialization of replacing traces that exhibits better performance than classical replacing trace schemes [16, p. 188]. The expression (16) shows the way in which this kind of traces are defined for the kNN-TD(λ) algorithm.

$$e(knn,j) = \begin{cases} p(knn), & j = a, \\ 0, & j \neq a, \end{cases} \tag{16}$$

where knn is the set of k-nearest neighbors of the observation s, $p(knn)$ are its corresponding probabilities and a is the last performed action. The traces always decay following the expression

$$e_{t+1} = \gamma\lambda e_t. \tag{17}$$

4. Continuous actions: the Ex$\langle a \rangle$(λ) algorithm

A pseudo-code of the Ex$\langle a \rangle$(λ) RL algorithm is presented in Algorithm 4. This algorithm is designed to deal with problems where the use of continuous actions have clear advantages over the use of fine grained discrete actions; some of the reasons of such advantage may include tractability–feasibility of the action space, learning speed and the accuracy of the approximated suboptimal actions with respect to the optimal actions.

Algorithm 4. The Ex$\langle a \rangle$(λ) reinforcement learning control algorithm for continuous state and action spaces.

```
1:  Initialize $\mathcal{A} \leftarrow linspace(lower, upper, n)$
2:  Initialize $\mathcal{L} \leftarrow 1, \ldots, n$
3:  Initialize the space of points $X \in R^n$
4:  Initialize Q arbitrarily and $e \leftarrow 0$
5:  repeat {for each episode}
6:      Initialize $s$, $e \leftarrow 0$
7:      $knn \leftarrow$ $k$-nearest neighbors of $s$
8:      $p(knn) \leftarrow$ probabilities of each $cl \in knn$
9:      $I \leftarrow \mathrm{argmax}_{\mathcal{L}}\ Q(knn,\mathcal{L})$
10:     $a \leftarrow \mathcal{A}[I] \cdot p(knn)$
11:     $\mathcal{Q}(knn,I) = \max_{\mathcal{L}} Q(knn,\mathcal{L}) \cdot p(knn)$
12:     repeat {for each step of episode}
13:         Take action $a$, observe $r$ and $s'$
14:         $knn' \leftarrow$ $k$-nearest neighbors of $s'$
15:         $p(knn') \leftarrow$ probabilities of each $cl \in knn'$
16:         $I' \leftarrow \mathrm{argmax}_{\mathcal{L}}\ Q(knn',\mathcal{L})$
17:         $a' \leftarrow \mathcal{A}[I'] \cdot p(knn')$
18:         $\mathcal{Q}(knn',I') = \max_{\mathcal{L}} Q(knn',\mathcal{L}) \cdot p(knn')$
19:         Update Q and e:
20:             $e(knn,\cdots) \leftarrow 0$ {optional}
21:             $e(knn,I) \leftarrow p(knn)$
22:             $\delta \leftarrow r + \gamma \mathcal{Q}(knn',I') - \mathcal{Q}(knn,I)$
23:             $Q \leftarrow Q + \alpha\delta e$
24:             $e \leftarrow \gamma\lambda e$
25:         $a \leftarrow a'$, $s \leftarrow s'$, $knn \leftarrow knn'$, $I \leftarrow I'$
26:     until $s$ is terminal
27: until learning ends
```

From the theoretical point of view, there are some ideas to understand what can be a continuous actions algorithm.

1. The first one is the "arbitrary points in the action space" which means that the algorithm is able (has the possibility) to produce

an action that translates into a particular point in the environment action space, e.g., an arbitrary point in the interval [0,1].

2. The second idea is the "continuous controller" point of view and has to do with the definition of a continuous function and is based on a simple concept: to a smooth variation in the environment state (s) corresponds a smooth variation in the emitted action (a). This idea is mathematically expressed as an interpretation of Cauchy's definition of a continuous function: a function f is continuous in the number s if f is defined in some open interval containing s and if for all $\varepsilon > 0$ exists a $\delta > 0$ such that:

$$\text{if } |s'-s| < \delta \text{ then } |f(s')-f(s)| < \varepsilon. \tag{18}$$

Thus, extending the definition to functions of several variables, we have that s and s' represent the state of the environment and that $f(s)$ and $f(s')$ represent the respective actions for each state of the environment, therefore: there will exist an arbitrary difference margin ε such that for environmental variations equal or less than $\delta(|s'-s| < \delta)$ the difference between the actions will be less than or equal to the margin ε $(|f(s')-f(s)| < \varepsilon)$.

3. Also, should a continuous reward function be considered so that a kind of gradient method could be used to find or refine the optimal action or policy?

As for the discrete actions method, kNN-TD(λ), an action list needs to be defined. Nevertheless the form and use of this action list will differ notoriously. When the actions are discrete these can have any type of nonnumeric interpretation, i.e. they can be symbols that indicate a label of an arbitrary action and these labels stored in the action list can be independent between them, whereas for continuous actions the action list is a granulation of a continuous space represented by points in a given interval, for instance:

$$\mathcal{A} = \{-1.0,-0.5,0.25,0.0,0.25,0.5,1.0\}.$$

In this case the actions are related to each other and are interdependent. As in the version for discrete actions, each neighbor will maintain an expectation $Q(a_i)$ for each action $a_i \in \mathcal{A}$.

The main modification to the method is made in the action selection mechanism and in the calculation of the value of the best action. That is, it is necessary to modify Eqs. (8) and (9). The action selection mechanism is now independent of the mechanism to calculate the value $\mathcal{Q}(knn',a')$ (the best prediction of future reward).

First, the action selection mechanism selects the recommended (best expected future reward) actions of each neighbor in the *knn* set and stores them in the optimal actions list A^*. Thus given an action list $\mathcal{A}$ the best actions are selected according to each neighbor:

$$I = \underset{a}{\operatorname{argmax}}\, Q(i,a) \quad \forall i \in [knn] \text{ and } \forall a \in \mathcal{A}, \tag{19}$$

where I is an index list over the action list $\mathcal{A}$ that will contain the indices of the values of $\mathcal{A}$ where the values of $Q(i,a)$ are maximal.

This index list I is then translated to its values within the list $\mathcal{A}$ to obtain the list A^* of the optimal values of action a recommended by each neighbor:

$$A^* = \mathcal{A}[I]. \tag{20}$$

Of this form the list A^* could contain e.g. the values $A^*=\{0.25,0.0,0.25,0.25\}$ for an index list $I=\{4,3,4,4\}$ and $k=4$.

Having the actions A^*, recommended by the neighbor in the *knn* set, and its respective probabilities given by $p(i)$ the *expected optimal action* is calculated according to the classical formulation of the expected value of a variable with discrete values. Thus the (collectively recommended) action to be performed by the agent can be obtained and calculated in the following way:

$$\langle a \rangle = \sum A^*(i)p(i) \quad \forall i \in knn, \tag{21}$$

where $\langle a \rangle$ is the expected value of the continuous optimal action to be executed, $A^*(i)$ is the best recommended action by the neighbor i and $p(i)$ is the probability $P(a=A^*(i)|s)$ that a takes the value $A^*(i)$ given the state of the environment s.

Finally, as in the kNN-TD algorithm, the update of the predictions of each neighbor is based on the use of *probability traces* and the TD-error (δ) to estimate the expected value of the future reward. Probability traces are defined in this case by the following equation:

$$e(knn,I) = \begin{cases} p(knn) & \text{for all } a \in I, \\ 0 & \text{for actions not in } I, \end{cases} \tag{22}$$

where $p(knn)$ are the probabilities of each neighbor in the *knn* and I is the index list over the actions in $\mathcal{A}$ with maximal action-value. Then the update rule (23) can be applied to each neighbor:

$$Q_{t+1} = Q_t + \alpha\delta e, \tag{23}$$

the TD-error (δ) can be calculated in the following form:

$$\delta = r + \gamma\mathcal{Q}(knn',I') - \mathcal{Q}(knn,I), \tag{24}$$

and the action-value $\mathcal{Q}(knn,I)$ is calculated according to the expression:

$$\langle \mathcal{Q}(knn,I) \rangle = \sum_{i=1}^{knn} \max_{\mathcal{L}} Q(i,\mathcal{L})p(i), \tag{25}$$

where $\mathcal{L}$ is the list $1\ldots n = |\mathcal{A}|$.

4.1. Explicit exploration

Although when initialized properly (e.g. optimistic) for a particular problem, TD methods perform an intrinsic exploration, the necessity of an explicit exploration/exploitation strategy it is always present due to, for instance, initial conditions disappear very fast or it is difficult to establish such initial conditions, but in general the most important reason for such an explicit exploration is the need to control and adapt the exploration/exploitation tradeoff.

Here we show a simple mechanism to include an $\varepsilon-$greedy exploration policy into the Ex$\langle a \rangle(\lambda)$ algorithm. There is just a little modification to the previous equations. The exploration mechanism works by obtaining an alternative form for Eq. (19) since this is the core of the action selection mechanism. In order to implement the $\varepsilon-$greedy strategy we have to use the equation.

First the *greedy* actions are defined:

$$greedy = \underset{a}{\operatorname{argmax}}\, Q(i,a) \quad \forall i \in [knn] \text{ and } \forall a \in \mathcal{A}, \tag{26}$$

where *greedy* is an index list on the action list $\mathcal{A}$ that will contain the indices of the values of A where the values of Q are maximal (i.e. the greedy actions) as in Eq. (19):

$$I_i = \begin{cases} greedy_i & \text{when rand}() > \varepsilon, \\ \text{random integer}(1,|\mathcal{A}|) & \text{otherwise}. \end{cases} \tag{27}$$

Then the index list I is translated to its values within the list $\mathcal{A}$ to obtain the list A^* as explained in Eqs. (20) and (21).

5. Handling noisy environments: the *k*-NNδs online adaptive filter

The k-NNδs is an online adaptive filter for noisy observations in continuous state spaces that learns of incremental form.

Since in some RL problems the observations are highly noisy, we have developed an online-adaptive filter based on the k-NN technique. This filter will learn the expected value of the state transition difference, thus the filter will learn based on a triplet state-action-nextstate (s,a,s') and for each pair (s,a) it must learn a delta (δ) which will represent the difference between the next state and the last state given the action a:

$$\langle \delta \rangle = s' - s|a, \tag{28}$$

thus it could be seen as that the algorithm estimates the expected value of the derivative of the systems state w.r.t. the action a:

$$\langle \delta \rangle = \frac{\partial s}{\partial a}. \tag{29}$$

Thus, the new state s_f will be

$$s_f = s + \langle \delta(s,a) \rangle. \tag{30}$$

The filter works with the same perceptual scheme as the k-NN perception exposed previously, that is, for each state s perceived it finds the knn set for the state s and its corresponding probabilities $p(knn)$. Then using the last action, a, performed it calculates a correction (filtering) of the s' state. A pseudo-code of the proposed filter in shown in Algorithm 5:

$$\Delta_s = (s' - s) - \delta(knn,a) \tag{31}$$

next, update the filter approximator

$$\langle \delta(knn,a) \rangle = \delta(knn,a) + \alpha \Delta_s p(knn) \tag{32}$$

finally, the filtered state s_f is obtained:

$$s_f = s + \sum_{i=1}^{knn'} \delta(i,a) p(i). \tag{33}$$

Algorithm 5. The k-NN δs online adaptive filter.

```
Require: $s,a,s'$
1:  Initialize $\delta(.,a)=0$ only in the first call.
2:  $knn \leftarrow$ $k$-nearest neighbors of $s$
3:  $p(knn) \leftarrow$ probabilities of each $cl \in knn$
4:  $\Delta s = (s' - s) - \delta(knn,a)$
5:  $\delta(knn,a) \leftarrow \delta(knn,a) + \alpha[\Delta s \cdot p(knn)]$
6:  return $s + \delta(knn,a) \cdot p(knn)$
```

6. Experimental results

The family of algorithms kNN-*TD* has been intensively tested on various kinds of problems with very robust results. These problems range from the classical Mountain-Car problem, The Acrobot and The Cart-Pole to some specific problems in robot control.

These algorithms (the kNN-TD(λ) methods) were rigorously tested at the Second and Third Annual Reinforcement Learning Competitions (RLC2008 and RCL2009) held in Helsinki and Montreal respectively [1,20], where the kNN-TD(λ) method (JAMH team) won in the PolyAthlon 2008 domain, obtained the second place in 2009 and also the second place in the Mountain-Car 2008 domain showing for two consecutive years that it is one of the state of the art general purpose reinforcement learning implementations. Also it got the second best mark on the Mountain-Car 2008 domain without a significant statistical difference with the best mark (a hand made no TD-Learning algorithm). These results are very significant since these test-problems were generalized and altered in very rare ways and with highly noisy state variables. In that problems we tested the kNN-TD(λ) algorithm with huge state spaces and more than 200 neighbors, i.e. a value of $k \geq 200$, while achieving good performance in computing cost terms. A key advantage of our approach was also the use of the k-NNδs filter.

We have also applied the Ex$\langle a \rangle(\lambda)$ reinforcement learning control algorithm for continuous state and action spaces presenting a method to obtain a near optimal RL controller [13] for the XCell Tempest RC Helicopter simulator created at Stanford University [2] and used also in the RLC2008 and RCL2009 competitions.

6.1. Mountain-Car

Fig. 2 shows the classical Mountain-Car control problem.

Here we present the result obtained for the Mountain-Car problem in order that it could be compared with the more standard results [16, p. 214] in the literature. In Fig. 3 the solution obtained

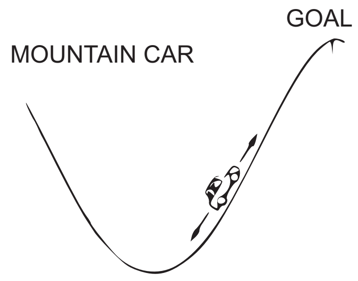

Fig. 2. A picture of the Mountain-Car problem.

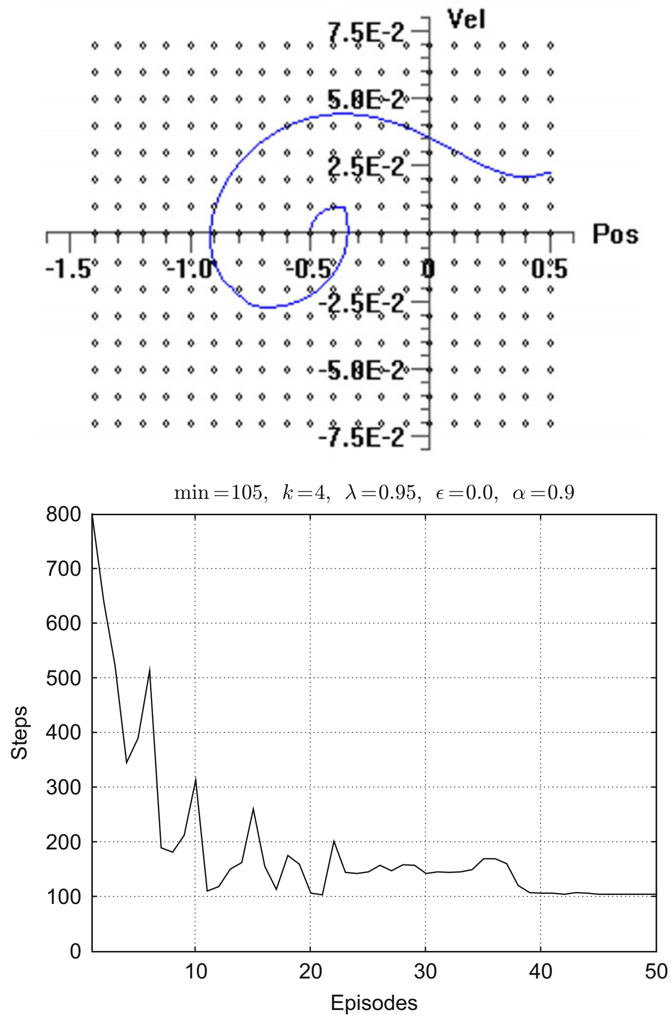

Fig. 3. State space and optimal solution for the Mountain-Car problem: the algorithm is stabilized from the episode 44 at 105 steps.

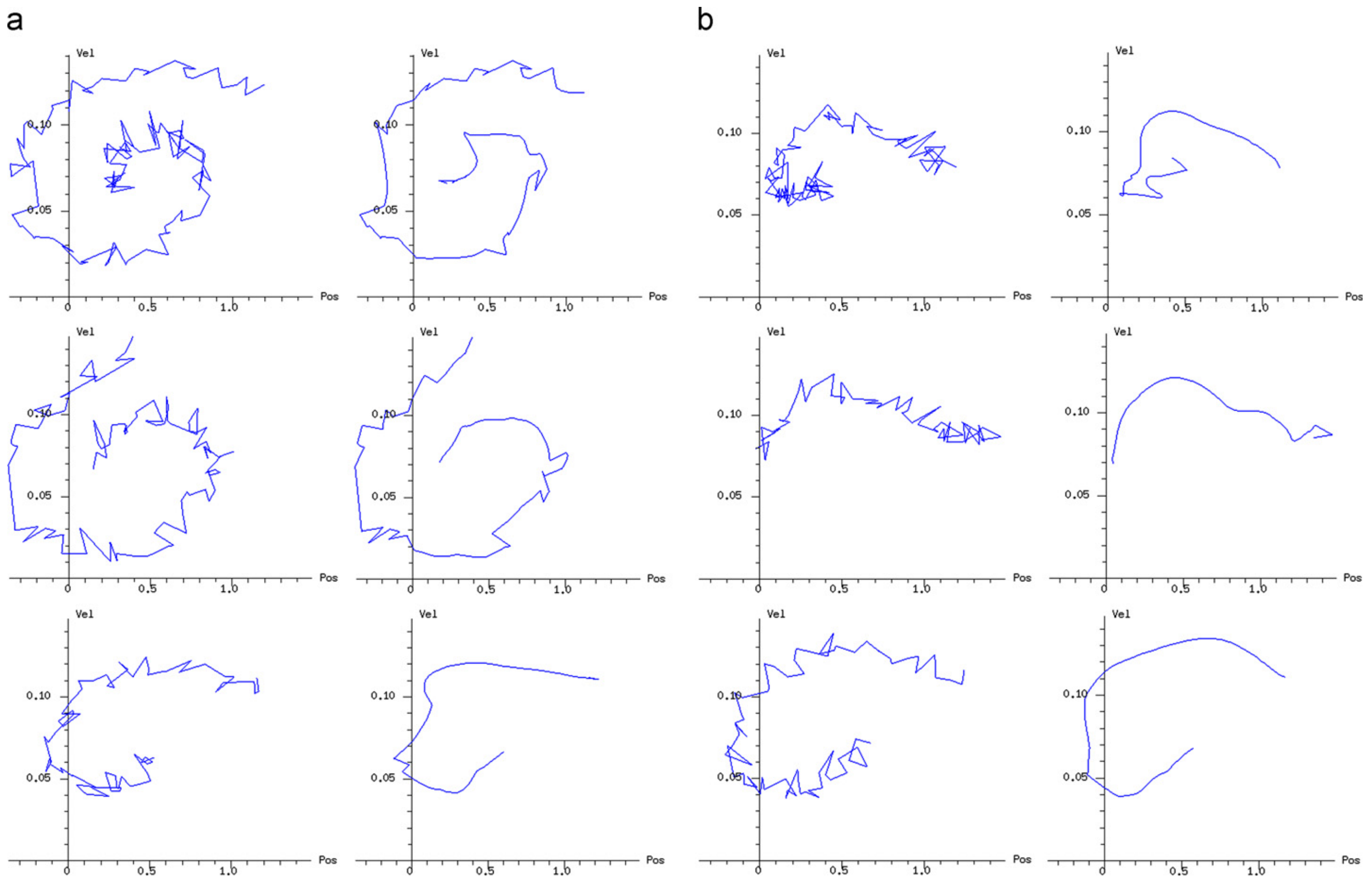

Fig. 4. (a) Filter behavior on the Mountain-Car Domain for several episodes during the RLC2008 Competition. In the initial episodes the filter has not learned yet. (b) Filter behavior on the Mountain-Car Domain for several episodes during the RLC2008 Competition. In the last episodes the filter has learned and thus it is able to reduce a very high amount of the noise in the input signal.

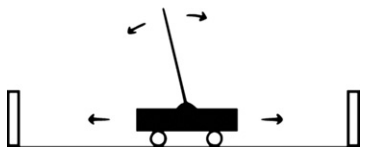

Fig. 5. The Cart-Pole balancing task.

by the kNN-TD(λ) algorithm is shown. This result was obtained with a value of $\lambda = 0.95$, $\alpha = 0.9$ and for a value of $k = 4$. Each dimension is defined by a set of points in the usual ranges with a linear partitioned space with 20 points for the position dimension and 15 points for the velocity dimension.

Figs. 4(a) and (b) show the performance of the k-NNδs online adaptive filter during the testing phase of the Second Annual Reinforcement Learning Competition (RLC2008). During the first episodes the filter is not adapted to the signal noise, however the algorithm converges quickly and the state signal becomes progressively noise free.

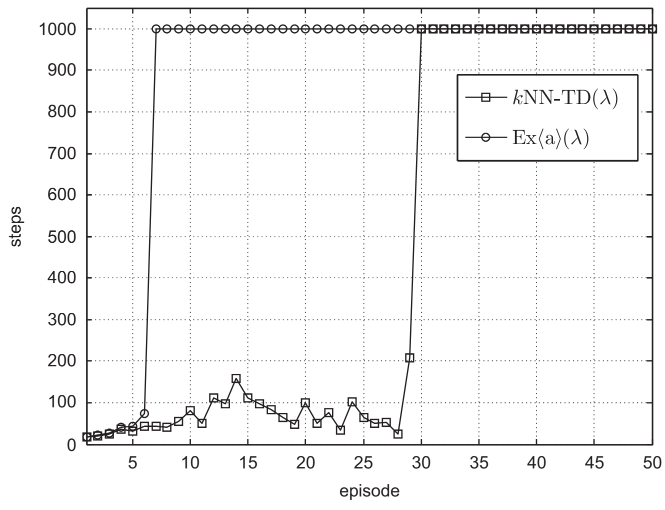

Fig. 6. Learning convergence curve for the discrete actions method kNN-TD(λ) ($k=4$; $\lambda = 0.95$; $\varepsilon = 0$; $\alpha = 0.3$) and for the continuous actions method Ex$\langle a \rangle$(λ) ($k=4$; $\lambda = 0.9$; $\varepsilon = 0$; $\alpha = 0.3$) in the Cart-Pole problem.

6.2. *Solving the Cart-Pole problem with continuous actions*

Fig. 5 shows a picture of the Cart-Pole balancing task (originally taken from the ANJI web site[1]).

The pole-balancing is a control benchmark historically used in engineering that is often used as an example of unstable, multiple-output and dynamic systems that appear in many balancing situations and it has been used to show modern and classical control-engineering techniques [3]. It involves a pole affixed to a

[1] http://anji.sourceforge.net/polebalance.htm

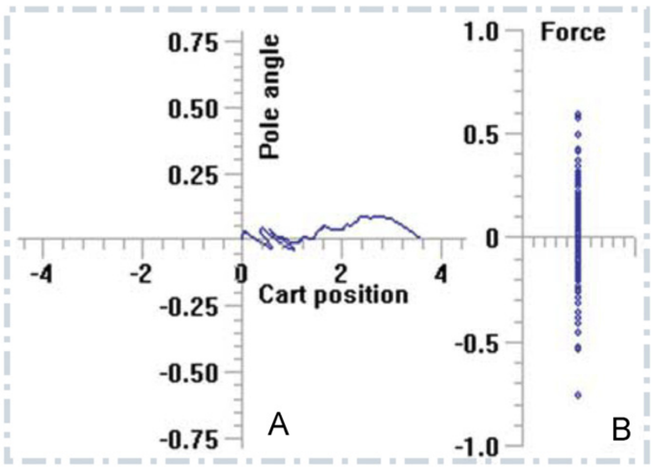

Fig. 7. Example episode of the Cart-Pole problem where the route in the state space in shown in (A) and the continuous performed actions are shown in (B). The value of the parameter k in this example was $k=8$ and the base action list was $\mathcal{A}=linspace(-1,1,11)$.

cart via a joint which allows movement along a single axis. The cart is able to move along a track of fixed length. Fig. 6 shows the learning convergence graph for the discrete actions method k-NN-TD(λ) in the Cart-Pole problem and the learning convergence graph for the continuous actions method Ex$\langle a \rangle$(λ). As one can see, the continuous actions algorithm outperforms the discrete version by learning a near optimal control policy in only five episodes while the discrete actions method takes at least 29 episodes in learning a near optimal policy. Also, when taking the softness of the control actions as a key factor, the quality of the control policies of the Ex$\langle a \rangle$(λ) algorithm can be considered better.

Finally in Fig. 7 a graph of an example episode where the route in the state space (A) and the continuous performed actions (B) can be seen. As it is observed in this example the actions that the algorithm performs are continuous and they almost completely cover the continuous action space.

7. Conclusions and further work

We have presented a series of robust high performance (award winning) implementations of reinforcement learning algorithms based on temporal-difference learning and weighted k-nearest neighbors. These algorithms were rigorously tested at the Second and Third Annual Reinforcement Learning Competitions (RLC2008 and RCL2009) held in Helsinki and Montreal respectively, where the kNN-TD(λ) method (JAMH team) won in the PolyAthlon 2008 domain, obtained the second place in 2009 and also the second place in the Mountain-Car 2008 domain showing that it is one of the state of the art general purpose reinforcement learning implementations. These algorithms are able to learn quickly, to generalize properly over continuous state spaces and also to be robust to a high degree of environmental noise. Also we presented the Ex$\langle a \rangle$ reinforcement learning algorithm for problems of continuous actions.

The perceptive process of the proposed methods was defined as a kind of k-nearest neighbors approach which produce a probabilistic representation of the input signal which constructs robust state descriptions based on the collection (knn) of receptive field units and the probability distribution vector $p(knn)$ over the knn collection.

The action selection mechanism of the proposed method can be understood, in k-NN terminology, as that the Agent will select the most "frequent" recommended action in the neighborhood of the observed state s or, in Bayesian terms, the action for which there is more evidence of a high future reward. The learning mechanism can be understood as a kind of temporal difference back-propagation which assigns the "right" TD error to each neighbor according to its respective influence in the decision making and action selection procedures, which is represented by its probability of being the nearest neighbor of the observed state.

Finally, immediate lines of future work can be identified, for instance adaptive state space determination, adaptive action space determination and, as a mandatory line of research, to provide improvements on the state space representations in order to find the k-nearest neighbors efficiently for avoiding the so-called "curse of dimensionality". Along the last line of future research, we can mention the promising techniques of approximate nearest neighbors search [10] which could lead to some compromise solutions between the accuracy of a k-NN expectations memory and its computational efficiency in complexity terms.

Acknowledgement

This work has been partially funded by the Spanish Ministry of Science and Technology, project DPI2006-15346-C03-02.

References

[1] The 2008 reinforcement learning competition, June 2008. URL http://2008.rl-competition.org/content/view/52/80/.
[2] P. Abbeel, V. Ganapathi, A.Y. Ng, Learning vehicular dynamics, with application to modeling helicopters, in: NIPS, 2005. URL http://books.nips.cc/papers/files/nips18/NIPS2005_0824.pdf.
[3] C.W. Anderson, Strategy learning in multilayer connectionist representations, in: Proceedings of the Fourth International Workshop on Machine Learning, Irvine, CA, Morgan KaufmannJune 1987, pp. 103–114 Morgan Kaufmann, 1987.
[4] C. Atkeson, A. Moore, S. Schaal, Locally weighted learning, AI Review 11 (April) (1997) 11–73.
[5] S. Bosman, Locally weighted approximations: yet another type of neural network, Master's Thesis, Intelligent Autonomous Systems Group, Department of Computer Science, University of Amsterdam, July 1996.
[6] T.M. Cover, P.E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory IT-13 (1) (1967) 21–27, ISSN 0018-9448.
[7] R.O. Duda, P.E. Hart, Pattern Classification and Scene Analysis, Wiley, 1973.
[8] S.A. Dudani, The distance-weighted k-nearest-neighbor rule, IEEE Transactions on Systems, Man and Cybernetics SMC-6 (4) (1976) 325–327.
[9] G.J. Gordon, Stable function approximation in dynamic programming, in: ICML, 1995, pp. 261–268.
[10] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: STOC, 1998, pp. 604–613. URL http://doi.acm.org/10.1145/276698.276876.
[11] L.P. Kaelbling, M.L. Littman, A.W. Moore, Reinforcement learning: a survey, Journal of Artificial Intelligence Research 4 (1996) 237–285 ISSN 1076-9757.
[12] J.A. Martin H, J. de Lope, A k-NN based perception scheme for reinforcement learning, in: EUROCAST, 2007, pp. 138–145.
[13] J.A. Martin H, J. de Lope, Ex$\langle a \rangle$: an effective algorithm for continuous actions reinforcement learning problems, in: Proceedings of 35th Annual Conference of the IEEE Industrial Electronics Society, Oporto, Portugal, IEEE Industrial Electronics Society, IEEENovember 2009, pp. 2063–2068 http://dx.doi.org/doi:10.1109/IECON.2009.5415084. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5415084.
[14] J.A. Martin H, J. de Lope Asiaín, D. Maravall, The kNN-TD reinforcement learning algorithm, IWINAC Part I, Methods and Models in Artificial and Natural Computation, Lecture Notes in Computer Science, Santiago de Compostela, Spain, vol. 5901, SpringerJune 2009, pp. 305–314, doi:10.1007/978-3-642-02264-7_32.
[15] S.P. Singh, R.S. Sutton, Reinforcement learning with replacing eligibility traces, Machine Learning 22 (1–3) (1996) 123–158.
[16] R. Sutton, A. Barto, Reinforcement Learning, An Introduction, The MIT Press, 1998 ISBN 0-262-19398-1.
[17] R.S. Sutton, Learning to predict by the method of temporal differences, Machine Learning 3 (1988) 9–44.
[18] R.S. Sutton, Rlai: reinforcement learning and artificial intelligence, 2006. URL http://rlai.cs.ualberta.ca/RLAI/rlai.html.
[19] C.J. Watkins, P. Dayan, Technical note Q-learning, Machine Learning 8 (1992) 279.
[20] D. Wingate, C. Diuk, L. Li, M. Taylor, J. Frank, Workshop summary: results of the 2009 reinforcement learning competition, in: ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning, New York, NY, USA, ACM, 2009, p. 1. ISBN 978-1-60558-516-1. http://doi.acm.org/10.1145/1553374.1553544.

Jose Antonio Martin H. received the B.S. and M.S. degrees in Computer Science from La Universidad del Zulia (LUZ) in 2002 and the Ph.D. degree in Computer Science from Universidad Politécnica de Madrid (UPM), Madrid, in 2009, "Studies on Adaptive Systems with applications in Autonomous Robots and Intelligent Agents." In addition, he is on the Ph.D. program of "Fundamentals of Basic psychology" at the Universidad Nacional de Educación a Distancia (UNED), Madrid, where he has received the Advanced Studies Diploma on "a computational model of the equivalence class formation psychological phenomenon." Since 2005, he is with the Department of Informatic Systems and Computing, Universidad Complutense de Madrid (UCM). His main research areas are cybernetics, machine learning and machine perception.

Javier de Lope (SM'94, M'98) received the M.Sc. in Computer Science from the Universidad Politécnica de Madrid in 1994 and the Ph.D. degree at the same university in 1998. Currently, he is Associate Professor in the Department of Applied Intelligent Systems at the Universidad Politécnica de Madrid. His current research interest is centered on the study, design and construction of modular robots and multi-robot systems, and in the development of control systems based on soft-computing techniques. He is currently leading a three-year R&D project for developing industrial robotics mechanisms which follow the guidelines of multi-robot systems and reconfigurable robotics. In the past he also worked on projects related to the computer-aided automatic driving by means of external cameras and range sensors and the design and control of humanoid and flying robots.

Darío Maravall (SM'78, M'80) received the M.Sc. in Telecommunication Engineering from the Universidad Politécnica de Madrid in 1978 and the Ph.D. degree at the same university in 1980. From 1980 to 1988 he was Associate Professor at the School of Telecommunication Engineering, Universidad Politecnica de Madrid. In 1988 he was promoted to Full Professor at the Faculty of Computer Science, Universidad Politecnica de Madrid. From 2000 to 2004 he was the Director of the Department of Artificial Intelligence of the Faculty of Computer Science at the Universidad Politecnica de Madrid. His current research interests include computer vision, autonomous robots and computational intelligence. He has published extensively on these subjects and has directed more than 20 funded projects,including a five year R&D project for the automated inspection of wooden pallets using computer vision techniques and robotic mechanisms, with several operating plants in a number of European countries. As a result of this project he holds a patent issued by the European Patent Office at The ague, The Netherlands.