The *k*NN-TD Reinforcement Learning Algorithm

José Antonio Martín H.[1], Javier de Lope[2], and Darío Maravall[2]

[1] Dep. Sistemas Informáticos y Computación, Universidad Complutense de Madrid
jamartinh@fdi.ucm.es
[2] Perception for Computers and Robots, Universidad Politécnica de Madrid
javier.delope@upm.es,dmaravall@fi.upm.es

Abstract. A reinforcement learning algorithm called kNN-TD is introduced. This algorithm has been developed using the classical formulation of temporal difference methods and a k-nearest neighbors scheme as its expectations memory. By means of this kind of memory the algorithm is able to generalize properly over continuous state spaces and also take benefits from collective action selection and learning processes. Furthermore, with the addition of probability traces, we obtain the kNN-TD(λ) algorithm which exhibits a state of the art performance. Finally the proposed algorithm has been tested on a series of well known reinforcement learning problems and also at the Second Annual RL Competition with excellent results.

1 Introduction

Reinforcement Learning (RL) [1] 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 which is used to evaluate the consequences of its actions. The framework of RL is designed to guide the learner in maximizing the average reward that it gathers from its environment in the long run.

Let us start with a classical Temporal Difference (TD) learning rule [2]:

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

where $x_t = r + \gamma \max_a Q(s_{t+1}, a)_t$.

This basic update rule of a TD-learning method can be derived directly from the formula of the expected value of a discrete variable. We know that the expected value (μ) of a discrete random variable x with possible values $x_1, x_2 \ldots x_n$ and probabilities represented by the function $p(x_i)$ can be calculated as:

$$\mu = \langle x \rangle = \sum_{i=1}^{n} x_i \, p(x_i), \tag{2}$$

thus for a discrete variable with only two possible values the formula becomes:

$$\mu = (1-\alpha)a \; + \; b\alpha \;\; , \tag{3}$$

J. Mira et al. (Eds.): IWINAC 2009, Part I, LNCS 5601, pp. 305–314, 2009.

where α is the probability of the second value (b). Then taking a as a previously stored expected value μ and b as a new observation x_t, we have:

$$\mu = (1 - \alpha)\mu \ + \ x_t\alpha \ , \tag{4}$$

and doing some operations we get:

$$\begin{aligned} \mu &= (1 - \alpha)\mu \ + \ x_t\alpha \\ &= \mu - \alpha\mu + \alpha x_t \\ &= \mu + \alpha[-\mu + x_t] \\ &= \mu + \alpha[x_t - \mu] \end{aligned} \tag{5}$$

Then we can see that the parameter α, usually described as the learning rate, express the probability that the random variable x get the value of the observation x_t and that replacing μ by $Q(s, a)_t$ and then replacing x_t by the expected cumulative future reward (this value is usually settled as the the learning target):

$$r_t + \gamma \max_a Q(s_{t+1}, a)_t \tag{6}$$

we finally get again the Q-Learning update rule (1).

In this paper we present a reinforcement learning algorithm called kNN-TD. This algorithm has been developed using the classical formulation of TD-Learning and a k-nearest neighbors scheme for the action selection process, expectations memory as well as for the learning process.

The k-nearest neighbors (k-NN) technique [3,4,5] 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.

As it has been proved [4] 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:

$$P(e_b|x) \leq P(e_{nn}|x) \leq 2P(e_b|x), \tag{7}$$

where $P(e_b|x)$ is the Bayes error rate and $P(e_{nn}|x)$ is the nearest neighbor error rate.

So we see that the upper bound on the probability of error for the NN-Rule is less than twice the Bayes error rate. Of this form we can note that the k-NN rule is an empirical approach to optimal decision making according to the Bayesian Theory and that a well established theoretical error bound is guaranteed.

There has been other uses of the k-NN rule, i.e. in Dynamic Programming (DP), as a function approximator with successful results [6]. Also, we must mention that the 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 [7]. Indeed LWL approximations can be viewed as a particular kind of artificial neural network (ANN) for approximating smooth functions [8]. While the survey by Atkeson and Moore [7] covers a wide range of characteristics and relevant parameters of LWL and Bosman [8] follows an approach to present LWL as a kind of neural network, we focus our approach in the relation between our methods and a probabilistic approach supported by the domain of non-parametric statistics and bayesian inference. Thus, there are several interpretations about these kinds of learning methods, each of them supported by some particular approach. Our approach has been developed independently starting with a preliminary work [9] proposing the use of k-NN as a perception scheme in RL.

Here we improve the initial proposed scheme by developing the kNN-TD(λ) which is a state of the art reinforcement learning algorithm. The algorithm is described in detail and a pseudo-code is presented. Also, public implementations of this algorithm will be published on the web site: `http://www.dia.fi.upm.es/~jamartin/download.htm` in order that these tools be used, evaluated or improved by the researchers interested in RL.

2 The kNN-TD Reinforcement Learning Algorithm

Figure 1 shows the agent-environment interaction cycle in RL. This interaction cycle could be described by four basic steps:

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.

The first task to address is the definition of the points (classifiers[1]) 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 over the

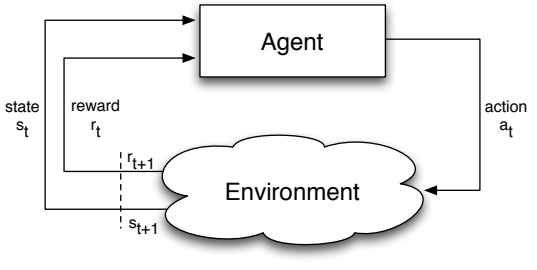


Fig. 1. Agent-Environment interaction cycle in RL

[1] Although the term experts is often used for the same purpose we prefer to use the term classifier which has more meaning in statistical machine learning.

complete state space. Despite this technique works for some problems it may happen that the absolute magnitudes of some dimensions be greater than others leading to a biased selection of the k-nearest neighbors over the dimension whose magnitude is smaller. For avoiding this potential bias it is necessary the correct standardization of all the dimensions in such a way that its absolute magnitudes don't biases the selection of the k-nearest neighbors. This standardization could be achieved by mapping all the points over a same range space, for instance, by applying the rule (8) which maps all points from the state space X to a space $\dot{x}$ in the range $[-1, 1]$:

$$\dot{x} = 2\left(\frac{x - \min(X)}{\max(X) - \min(X)}\right) - 1, \tag{8}$$

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

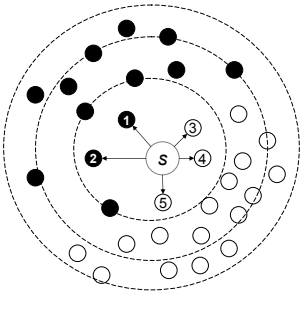
Finally for each defined classifier cl_i in the state space will be an associated action-value predictor $Q(i, a)$.

2.1 Perception

The first task is to determine the k-nearest neighbors set (knn). Figure 2 illustrates a time step of the perceptual process for a value of $k = 5$ and the 5-nearest neighbors of an observation s. Columns $Q(a1)$ and $Q(a2)$ of figure 2 represent the reward prediction for each action (1 or 2) for each classifier $c_i \in [1, ..., 5]$, column d is the euclidian distance from the current mapped observed point s to each classifier c_i and the values of column w represents the weights of each classifier based on the euclidian distance following the formula:

$$w_i = \frac{1}{1 + {d_i}^2} \quad \forall i \in [1, ..., k] \tag{9}$$

Thus less distance imply more weight (activation) and then for each time step will be k active classifiers whose weights are inverse to the euclidian distance from the current observation (s).



	$Q_{(a1)}$	$Q_{(a2)}$	d	w	$V_{(a1)}$	$V_{(a2)}$
❶	1.00	-1.00	1.75	0.25	0.25	-0.25
❷	2.00	-2.00	2.50	0.14	0.28	-0.28
③	0.50	0.75	1.00	0.50	0.25	0.38
④	-2.00	2.75	1.00	0.50	-1.00	1.38
⑤	-1.00	1.13	1.50	0.31	-0.31	0.35
					-0.54	1.58

Fig. 2. The k-nearest neighbors rule at some time step for a value of $k = 5$

Then we can state that a perceptual representation, or simply a perception, is uniquely and completely defined by two components (knn, w) obtained from a currently observed state (s):

1. A set (knn) which contains the k-nearest neighbors of the observation s.
2. A vector (w) which contains the weights (or activations) of each element in the knn set.

2.2 Action Selection

Once obtained a perceptual representation (knn, w) 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 expectations about the consequences of such action (a). These consequences are represented by expectations of future rewards. The discussion about exploration vs. exploitation and the corresponding methods like greedy, ϵ-greedy or soft-max methods can be found in the literature [1]. We will center the attention on calculating the expectations 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 consist 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 set, the procedure gets reduced to estimate an expected value, that is:

$$\mu = \sum_{i=1}^{n} x_i \; p(x_i) \; , \tag{10}$$

In this case the probabilities of each value are implied by the current weights vector (w). This vector should be normalized in order to express a probability distribution $p(.)$, thus the probabilities for each element in the knn set are calculated by the equation (11).

$$p(i) = \frac{w_i}{\sum w_i} \quad \forall i \in knn, \tag{11}$$

In this way, the expected value of the future reward for a given action a is estimated by (12).

$$\langle V(a) \rangle = \sum_{i=1}^{knn} Q(i,a) p(i) \; , \tag{12}$$

hence $p(i)$ is the probability $P(Q(i,a) = V(a)|s)$ that $V(a)$ takes the value $Q(i,a)$ given the state of the environment s. Then the action selection mechanism can directly derive a greedy policy from the expectations for each perceptual representation as shown in (13).

$$\max V(a) \tag{13}$$

This 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.

2.3 Learning

The learning process involves the use of the accumulated experience and thus it will rely on past and current experienced states, actions performed and received rewards (s, a, s' and r). Since we restrict the RL problem to a Markov Decision Process (MDP) we only need to deal with the immediate experienced states and performed actions.

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, once for each state respectively. These perceptual representations are the tuples (knn, w) for state s and (knn', w') for state s'.

Following the TD-Learning scheme we need an estimation of the future reward. This estimation is generally obtained by the value of the current state s' represented by $\max_a Q(s', a)$ in off-policy learning methods and by $Q(s', a')$ in on-policy learning.

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

$$\langle V'(a) \rangle = \sum_{i=1}^{knn'} Q(i, a) p'(i) \ , \tag{14}$$

where knn' is the set of the k-nearest neighbors of s', $p'(i)$ are the probabilities of each neighbor in the set knn' and the TD error (δ) can be calculated as shown in (15) or (16).

$$\delta = r + \gamma V'(a') - V(a) \tag{15}$$

$$\delta = r + \gamma \max_a V'(a) - V(a) \tag{16}$$

Thus we can simply update the prediction of each point in the knn set by applying the update rule (17).

$$Q(i, a)_{t+1} = Q(i, a)_t + \alpha \delta p(i) \ \ \forall i \in knn \tag{17}$$

2.4 Probability Traces $e(i) \leftarrow p(i)$: The k-NN$Q(\lambda)$ Algorithm

The use of eligibility traces [1, 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 behaves between two extremes: one-step TD-Learning ($\lambda = 0$) and Monte Carlo methods ($\lambda = 1$) [1, p.163]. A pseudo-code of the kNN-TD(λ) RL algorithm is presented in algorithm 1.

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

$$Q_{t+1} = Q_t + \alpha \delta e \tag{18}$$

Algorithm 1. The kNN-TD(λ) reinforcement learning control algorithm

Initialize the space of classifiers cl
Initialize $Q(cl, a)$ arbitrarily and $e_i(cl, a) \leftarrow 0$
repeat {for each episode}
  Initialize s
  $knn \leftarrow k$-nearest neighbors of s
  $p(knn) \leftarrow$ probabilities of each $cl \in knn$
  $V(a) \leftarrow Q(knn, a) \cdot p(knn)$ for all a
  Chose a from s according to $V(a)$
  repeat {for each step of episode}
    Take action a, observe r, s'
    $knn' \leftarrow k$-nearest neighbors of s'
    $p(knn') \leftarrow$ probabilities of each $cl \in knn'$
    $V'(a) \leftarrow Q(knn', a) \cdot p(knn')$ for all a
    Chose a' from s' according to $V'(a)$
    Update Q and e:
      $e(knn, \cdots) \leftarrow 0$ {optional}
      $e(knn, a) \leftarrow p(knn)$
      $\delta \leftarrow r + \gamma V'(a') - V(a)$
      $Q \leftarrow Q + \alpha\delta e$
      $e \leftarrow \gamma\lambda e$
    $a \leftarrow a'$, $s \leftarrow s'$, $knn \leftarrow knn'$, $p \leftarrow p'$
  until s is terminal
until learning ends

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

Cumulating traces are defined in a simple form:

$$e(knn, a)_{t+1} = e(knn, a)_t + p(knn), \tag{19}$$

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 [10,1]. Replacing traces in the kNN-TD(λ) algorithm are defined in a simple way by establishing the value of the trace for each classifier 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 traces schemes [1, p.188]. The expression (20) shows the way in which this kind of traces are defined for the kNN-TD(λ) algorithm.

$$e(knn, j) = \begin{cases} p(knn) & j = a \\ 0 & j \neq a \end{cases} \tag{20}$$

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 (21).

$$e_{t+1} = \gamma\lambda e_t \tag{21}$$

As we can see, the eligibility traces scheme is replaced by a probability traces scheme given the fact that the stored values are just probabilities and not the value 1 as is used in classical eligibility traces theory.

3 Performance and Evaluation

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

Figure 3 shows the graphical representation of two classical control problems: the Mountain-Car problem (taken from [1]) and the Cart-Pole balancing task (taken from the ANJI web site `http://anji.sourceforge.net/`).

First we present the result obtained for the Mountain-Car problem in order that it could be compared with the more standard results [1, p.214] in the literature.

In figure 4 the final solution obtained 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.

Figure 3 shows the learning convergence graph for method kNN-TD(λ) in the Cart-Pole problem. As can be seen the algorithm solves the problem in only 29 episodes.

Even more, this algorithm was tested at the Second Annual RL Competition hosted at `http://www.rl-competition.org/` (RL2008) with excellent results. The kNN-TD(λ) algorithm won the PolyAthlon event (JAMH team) showing that is a method of general applicability as well as of high performance. Also it got the second best mark on the Mountain-Car domain without a significant statistical difference with the best mark. This result is very significative since this problems where generalized and altered in rare ways indeed with a highly noisy state variables. In that problems we tested the kNN-TD(λ) algorithm with a huge state space and more than 200 nearest-neighbors, that is with a value of $k \geq 200$, while achieving good performance in computing cost terms.

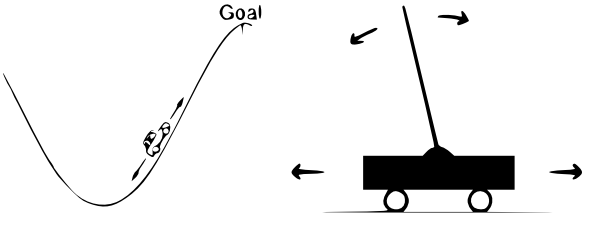


Fig. 3. The Mountain-Car problem and the Cart-Pole balancing task

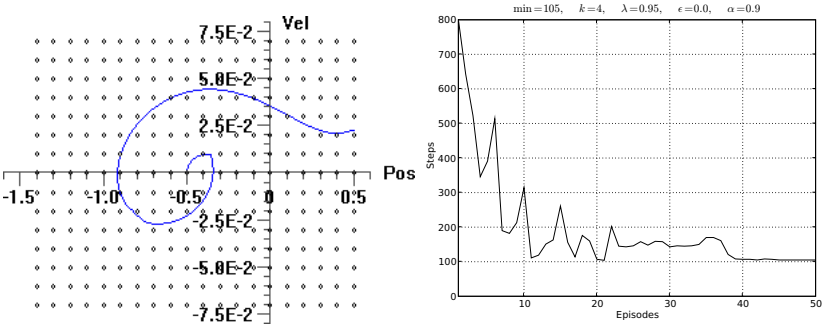


Fig. 4. State space and optimal solution for the Mountain-Car problem: the algorithm is stabilized from the episode 44 at 105 steps

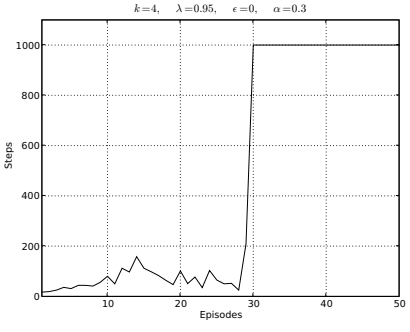


Fig. 5. Learning convergence graph for the method kNN-TD(λ) in the Cart-Pole problem

4 Conclusions and Further Work

We have introduced a reinforcement learning algorithm called kNN-TD. We have extended the basic algorithm with a technique called probability traces, which is a natural specialization of eligibility traces in this context, obtaining the kNN-TD(λ) algorithm. This new algorithm can be compared with the state of the art reinforcement learning algorithms both in performance and simplicity terms as shown by the experimental results.

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 the learning mechanism can be understood as a kind of temporal difference back-propagation which assigns the right TD error to each classifier 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.

Finally, immediate lines of future work can be identified, for instance adaptive classifier 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 [11] which could lead to some compromise solutions between the accuracy of a k-NN expectations memory and its computational efficiency in complexity terms.

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

References

1. Sutton, R., Barto, A.: Reinforcement Learning, An Introduction. MIT Press, Cambridge (1998)
2. Watkins, C.J., Dayan, P.: Technical note Q-learning. Machine Learning 8, 279 (1992)
3. Cover, T.M., Hart, P.E.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory IT-13(1), 21–27 (1967)
4. Duda, R.O., Hart, P.E.: Pattern Classification and Scene Analysis. Wiley, Chichester (1973)
5. Dudani, S.A.: The distance-weighted k-nearest-neighbor rule. IEEE Transactions on Systems, Man and Cybernetics SMC-6(4), 325–327 (1976)
6. Gordon, G.J.: Stable function approximation in dynamic programming. In: ICML, pp. 261–268 (1995)
7. Atkeson, C., Moore, A., Schaal, S.: Locally weighted learning. AI Review 11, 11–73 (1997)
8. Bosman, S.: Locally weighted approximations: yet another type of neural network. Master's thesis, Intelligent Autonomous Systems Group, Dep. of Computer Science, University of Amsterdam (July 1996)
9. Martin, H., Antonio, J., de Lope, J.: A k-NN based perception scheme for reinforcement learning. In: Moreno Díaz, R., Pichler, F., Quesada Arencibia, A. (eds.) EUROCAST 2007. LNCS, vol. 4739, pp. 138–145. Springer, Heidelberg (2007)
10. Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Machine Learning 22(1-3), 123–158 (1996)
11. Indyk, P., Motwani, R.: Approximate nearest neighbors: Towards removing the curse of dimensionality. In: STOC, pp. 604–613 (1998)