On Determinism Handling While Learning Reduced State Space Representations

Fernando Fernández[1] and **Daniel Borrajo**[2]

Abstract. When applying a Reinforcement Learning technique to problems with continuous or very large state spaces, some kind of generalization is required. In the bibliography, two main approaches can be found. On one hand, the generalization problem can be defined as an approximation problem of the continuous value function, typically solved with neural networks. On the other hand, other approaches discretize or cluster the states of the original state space to achieve a reduced one in order to learn a discrete value table. However, both methods have disadvantages, like the introduction of non-determinism in the discretizations, parameters hard to tune by the user, or the use of a high number of resources. In this paper, we use some characteristics of both approaches to achieve state space representations that allow to approximate the value function in deterministic reinforcement learning problems. The method clusters the domain supervised by the value function being learned to avoid the non-determinism introduction. At the same time, the size of the new representation stays small and it is automatically computed. Experiments show improvements over other approaches such as uniform or unsupervised clustering.

1 INTRODUCTION

Reinforcement learning algorithms have problems when it is applied to large or continuous state spaces. Given that a value of "utility" must be computed for each state and action in the domain, generalization techniques have to be used to avoid an inefficient use of the experience. On one hand, some generalization techniques rely on the use of neural networks [3] to approximate the value function used in most of the reinforcement learning algorithms [2]. Several comparisons can be found in the literature [4, 14]. However, the use of neural networks introduces several problems. For instance, in [4], convergence properties of several neural networks are studied, showing that convergence is not always ensured. Furthermore, it is known that it is not always easy to define the right arquitecture of the neural network to achieve a successful solution.

On the other hand, generalization techniques such as CMAC [16] or variable resolution discretization [13] try to map different areas of the domain that group several states (even infinite continuous states) to a same state from a finite discretized state space representation. The way to represent the new discretized states is diverse, and range from coarse coding [9], tiling [1], radial basis functions [5], vector quantization approaches [7] to self organizing maps [10]. However, these kinds of representations have similar problems to the previous approach. For instance, all of them require to define the resources used (number of regions, centroids, etc.). Furthermore, they can convert deterministic environments into stochastic ones by grouping in the same cluster examples of different classes (best action to apply). Instance based approaches solve some of the previous problems, but they do not work well when the number of instances grows. Since they require an efficient matching function, one solution consists of building a kd-tree that organises the instances [15]. This organization is a form of clustering the instances, and therefore it can be considered as an implicit discretization.

In this work, we have focused on finding state space representations that allow to keep (or loose as little as possible) the determinism of the domains. To do this, we study the use of an algorithm that allows to learn both the state space representation and the value function at the same time. The discretization method is based on a nearest prototype approach, that defines the regions of the environment by prototypes and the nearest neighbour rule. Furthermore, we wanted that the number of prototypes (i.e. the number of regions) be small and automatically adapted to the requirements of the learning of the value function.

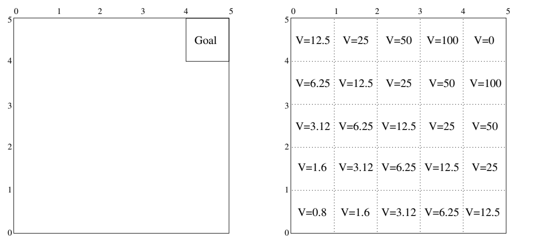
The next section introduces a deterministic domain that has a continuous state space. In that section, how typical discretization techniques can introduce non-determinism will be shown, even when the original domain was deterministic. Then, the proposed approach will be described, followed by the presentation of the experiments performed with different techniques and some comparative results. Lastly, the main conclusions obtained will be presented.

2 DISCRETIZATION APPROACHES FOR AN EXPLORATION DOMAIN

The classical exploration domain or grid world, as shown in Figure 1(a), is a domain where a robot has to learn to arrive to the goal area from random initial positions. For clarifying reasons, we will use a simplified version where actions of the robots are limited to follow cardinal points, i.e. "go East", "go North", "go West" and "go South". The area size is 5×5, and all the actions produce a motion of size one. If the robot tries to execute an action that should end out of the arena, the robot is forced to stay in the original position, and we will call it a blocked situation.

If we use a reinforcement function that returns a maximum reward of 100 when the robot arrives to the goal area and 0 otherwise, and we use a discount factor of $\gamma = 0.5$, for each executed action, the optimal value function is shown in Figure 1(b). At the same time, the limits of the value function present an optimal state space representation that does not introduce non-determinism. The limits of the different regions are defined by the discontinuities of the value

[1] Universidad Carlos III de Madrid, Avda/ de la Universidad 30. 28911, Leganés, Madrid, Spain. E-mail: ffernand@inf.uc3m.es
[2] Universidad Carlos III de Madrid, Avda/ de la Universidad 30. 28911, Leganés, Madrid, Spain. E-mail: dborrajo@ia.uc3m.es



(a) Grid-World Domain (b) Optimal Value Function

Figure 1. Grid-world domain. (a) shows the goal area and (b) the optimal value function for $\gamma = 0.5$.

function, so in this case, only 25 discretized states are needed.

Although this is a deterministic domain, it is easy to show that obtaining a state space representation that keeps this determinism in the environment without the use of any prior knowledge is not an easy task. For instance, since "a priori" we might not know the number and disposition of the regions, imagine that instead of introducing 5x5 regions, we introduce 6x6 regions. The result is illustrated in Figure 2. The environment introduces a high non-determinism because, even if the execution of the same action from the same state always achieves the same final state, in some executions the goal could be achieved, while not in others. So immediate rewards can be different in different executions, resulting in a non-deterministic behaviour.

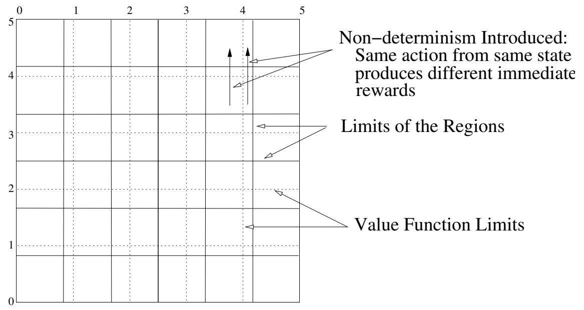


Figure 2. Example of uniform discretization with 6×6 regions in the grid-world domain.

3 LEARNING THE STATE SPACE REPRESENTATION AND THE Q FUNCTION

Summarizing the problems presented above, we can say that the goals of our desired state space representation are to keep the determinism of the domain, that the number of states does not grow much, and that the user does not have to introduce "a priori" information. Given that we have a deterministic environment, we know there will be a finite set of values that the value function can achieve, although we do not know "a priori" which ones they are. For instance, in Figure 1(b), if we knew the optimal discretization, the optimal value function would only take 9 different values. These values can be described as $r_g * \gamma^i$, for $i = 0, \ldots, p-1$, where r_g is the reward received when achieving the goal (in this case, $r_g = 100$), and p is the longest optimal path between any initial point of the environment and the goal (in this case, $p = 8$). Therefore, if we consider each of those values as a different class, we can formulate the problem of discretizing the domain as a supervised learning problem in which the state descriptions are mapped into attributes (x and y in this case), and the Q values are mapped into the classes (9 classes in this case). As we have said before, these classes are not known "a priori", so our goal is that they progressively appear by applying Q-Learning over the experience tuples.

However, we are not still done. We need to output the set of new states, such that the non-determinism does not grow when using those new states for computing the Q function. Therefore, the classifier algorithm to use should not only provide a successful classification of the Q values for a given state, but also the new state space representation. As an example, nearest neighbour/prototype classifiers satisfy this condition, in the sense that they typically achieve good accuracy, at the same time that they generate a set of prototypes that can be used as a state space representation. So, both the state space representation and the value function are learned at the same time, because they are inherently related.

All these ideas are introduced in a new discretization technique called ENNC-QL that is described next. It uses a nearest neighbour algorithm, ENNC, to learn the state space representation supervising it with the Q values, while it uses the *Q-Learning* update function to compute these Q values. Next section introduces the nearest neighbour algorithm, called *Evolutionary Nearest Neighbour Classifier* (ENNC).

3.1 The ENNC algorithm

1-Nearest Neighbour Classifiers (1-NN) are a particular case of k-NN classifiers that assign to each new unlabelled example, e, the label of the nearest prototype c from a set of N different prototypes previously classified [6]. The main generic parameters of this sort of classifiers are: the number of prototypes to use, their initial position, and a smoothing parameter. The ENNC algorithm [8] is a 1-nearest neighbour classifier whose main characteristic is that it has a small number of user defined parameters: it only needs the number of iterations to run. This means that none of the above parameters has to be supplied.

The algorithm starts with only one prototype in the initial set of prototypes. This set is modified iteratively by the execution of several operators in each execution cycle. At the end of the evolution, the classifier is composed of a reduced set of prototypes correctly labelled. The main steps of the algorithm are summarized next:

- Initialization. The initial number of prototypes is one. The method is able to generate new prototypes stabilizing in the most appropriate number in terms of a defined "quality" measure.
- Execute N cycles. In each cycle, execute the following operators:

  Information Gathering. At the beginning of each cycle, the algorithm computes the information required to execute the operators. This information relates to the prototypes, their quality and their relationship with existing classes.

  Labelling. It labels each prototype with the most popular class in its region.

  Reproduction. It introduces new prototypes in the classifier. The insertion of new prototypes is a decision that is taken by each prototype; each prototype has the opportunity of introducing a new prototype in order to increase its own quality.

  Fight. It provides each prototype the capability of getting training patterns from other regions.

  Move. It implies the reallocation of each prototype in the best expected place. This best place is the centroid of all the training patterns that it represents.

  Die. It eliminates prototypes. The probability to die is inversally proportional to the quality of each prototype.

3.2 The ENNC-QL Algorithm

We define the learning problem in ENNC-QL as follows. Given a domain with a continuous state space, where an agent can execute a set of L actions $A = \{a_1, \ldots, a_L\}$, the goal is to obtain an approximation of the action value function, $Q(s, a)$. In this case, and following other ideas from neural networks approaches as [11], one function approximator is used for each action to represent the value of applying the action in each state. I.e. we need L function approximators $Q_{a_i}(s)$, for $i = 1, \ldots, L$.

The main idea of this approach is that given L function approximators, L different state space representations, $S_{a_i}(s)$, will be computed, because each one may require different resources. Each $S_{a_i}(s)$ is a function that takes as input a state of the original representation, and outputs a state in the new representation. The approximators $Q_{a_i}(s)$ are learned at the same time, so at the end of the algorithm we have both the L state space representations and the approximation of the Q function for each action. It does so by propagating the rewards from the goal areas to the rest of the environment, and learning the Q value step by step with the *Q-Learning* update function for deterministic domains (equation 1 with α set to 1).

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha\{r + \gamma \max_{a'} Q(s', a')\} \quad (1)$$

A high level description of the algorithm is shown in Figure 3. The algorithm starts with an initialization step, where the L $S_{a_i}(s)$ state space representations are initialized to only one state, and the L $Q_{a_i}(s)$ approximators are initialized, typically to 0.

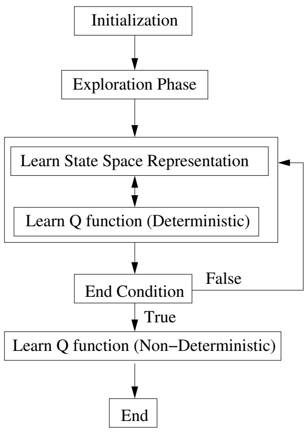


Figure 3. High level description of the ENNC-QL algorithm.

The second step is an exploratory phase. This phase generates a set T of experience tuples, of the type $< s, a_i, s', r >$, where s is any state, a_i is the action executed from that state, s' is the state achieved and r is the immediate reward received from the environment. From this initial set of tuples and using S_{a_i} and Q_{a_i}, $i = 1, \ldots, L$, the Q-learning update rule for deterministic domains can be used to obtain L training sets, T_i^0, $i = 1, \ldots, L$, with entries of the kind $< s, c_{s,a_i} >$ where c_{s,a_i} is the resulting value of applying the Q update function to each training tuple. In this first step, $Q_{a_i} = 0$, $i = 1, \ldots, L$, for all s, so the possible values for c_{s,a_i} depend only on the possible values of r. If we suppose the r values are always 0, except when a goal state is achieved, where a maximum reward of 100 is obtained, the only two values for c_{s,a_i} are 0 and 100 in the first iteration.

Thus, we have L sets of pairs $< s, c_{s,a_i} >$ that can be used by a supervised learning technique to learn L approximators, one for each action. We will focus now in only one of them. If we use a nearest neighbour technique such as ENNC as a supervised learning approach, the algorithm will return two things. The first one, a good approximation of the action-value function. The second one, an efficient state space representation for approximating the value function, given that nearest neighbour classifiers return a set of prototypes that defines a set of regions in the state space. This process is executed iteratively k times, that is supposed to be big enough to learn the whole state space.

The resulting state space representation is deterministic if: the original domain is deterministic; and the classifier does not introduce non-determinism (it achieves a 100% of accuracy). However, given that the classifier algorithm can introduce errors and it might not be able to successfully classify the 100% of the data in each iteration, the resulting state space representation will still have some non-determinism. We apply, then, a new learning step to absorb as much as possible this non-determinism. We use the update function of *Q-Learning* for stochastic domains, that is shown in equation 1, and we learn over the resulting state space representation. If the classifier system achieves a 100% of success in all classifications, this last step could be omitted. The algorithm is formalized in Figure 4.

ENNC-QL Algorithm

- Obtain by exploration a set T of state transitions $< s, a_i, s', r >$
- Set $t = 0$ and s_0 to any state
- Initialize the state space representation for each action a_i and each state s_j: $S^t_{a_i}(s_j) = \{s_0\}$
- Initialize the initial approximators for each action a_i: $Q^t_{a_i}(s_0) = 0$
- Repeat k times
  - Obtain for each action a_i the new representation $S^{t+1}_{a_i}$, and the approximator $Q^{t+1}_{a_i}(s)$:
    - ∗ Generate a training set, T^t_i, for the classifier. For each tuple $< s, a_i, s', r >$ in T:
      - · Compute the new class value for s and a_i:

$$c_{s,a_i} = r + \gamma \max_{a_j \in A} Q^t_{a_j}(s') \quad (2)$$

      - · Introduce the pair $< s, c_{s,a_i} >$ in the training set, T^t_i.
    - ∗ Apply the ENNC algorithm over the training set T^t_i, using as seed $S^t_{a_i}$. The resulting state space representation is $S^{t+1}_{a_i}$ and the resulting approximator is $Q^{t+1}_{a_i}(s_{ij})$, for all $s_{ij} \in S^{t+1}_{a_i}$.
  - Set $t = t + 1$
- Use the non-deterministic Q-Learning update function to learn the final $Q_{a_i}(s)$ functions, when the classifier system does not achieve the 100% of classification accuracy.

Figure 4. The ENNC-QL algorithm.

A characteristic of the method is that each state space representation $S^{t+1}_{a_i}$ is very similar to its previous one, $S^t_{a_i}$. Thus, to learn the state space representation in each iteration, the state space representation learned in the previous iteration can be used as seed, improving the performance of the learning technique.

4 EXPERIMENTS AND RESULTS

To test the performance of the method, we have used the domain shown in Figure 5, that consists in a robot moving into an office area. This area is represented by walls, free positions and goal areas, all of them of size 1×1. Thus, the whole domain is 24×21. The possible actions that the robot can execute are “go North”, “go East”, “go South” and “go West”, all them of size 1. The robot is supposed to know its location in the space by the coordinates (x, y) provided by some localization system. Furthermore, the robot has an obstacle avoidance system that blocks the execution of actions if they will make the robot crush into a wall. Two goal areas have been located in two different rooms.

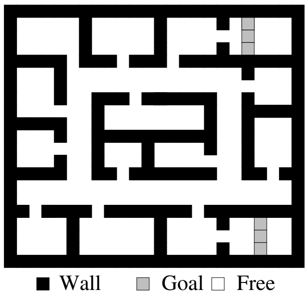


Figure 5. Office Domain.

We have performed several experiments, comparing uniform discretization and VQ discretizations of different sizes with the only solution reported by ENNC-QL. Training data is obtained from 4,000 runs. In each run, the robot tries to achieve the goal area from random initial positions. Each run is limited to the execution of 10 actions. For comparison reasons, we first made some experiments with uniform discretizations of the environment and with state space representations learned with the generalized Lloyd algorithm [12]. Experiments were performed over different state space sizes. Figure 6 shows a uniform discretization of the domain with 1024 states, while Figure 7 shows a discretization computed with the Generalized Lloyd Algorith, of the same size. The last one locates prototypes only in relevant positions of the domain.

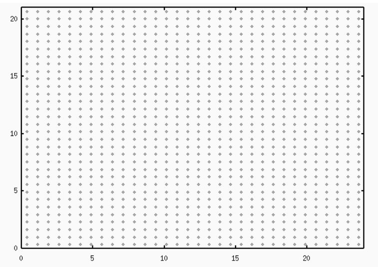


Figure 6. Prototypes Using Uniform Discretization.

The *Q-Learning* update function shown in Equation 1 was used for learning the policy, using $\gamma = 0.9$ and a constant value of $\alpha = 0.1$ (values selected experimentally). Results are summarized in Figure 8, for tests of 4000 runs. In each test, the robot is located in a random position, and it tries to achieve the goal by executing the optimal actions according to the Q function learned with the representations generated by each discretization technique. The maximum length of a run is 100 actions. If the robot does not achieve the goal area after executing 100 actions, the robot is supposed to be in a blocked situation or a cycle.

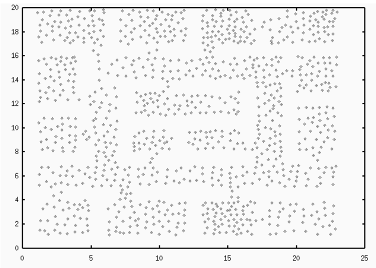


Figure 7. Prototypes Using VQ Discretization.

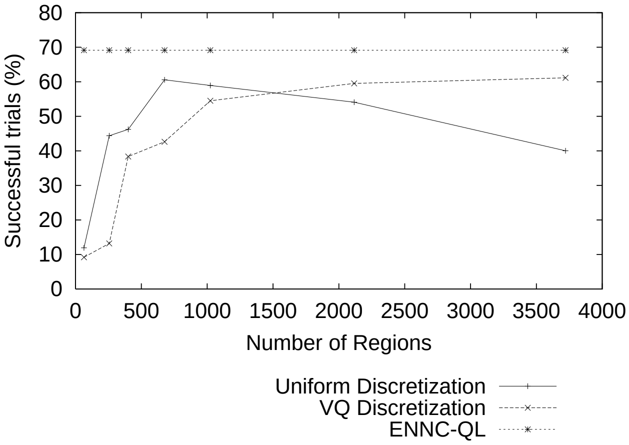


Figure 8. Percentage of successful trials when learning with different approaches and different state space sizes.

From these results one can draw the following conclusions. Uniform discretizations requires a higher number of states to generate good results. In this case, the 61.15% of successful trials is only achived with 3721 states. However, and given the regularities of this domain, with a uniform discretization of 24×21 states, the 100% of success could be achieved (in the same way that the 5×5 discretization for the domain presented in section 2). On the other hand, the VQ discretization takes advantage of the best location of the prototypes, achieving the 60.57 of successful trials with 676 states. When the number of states is increased, using VQ discretizations the performance decreases too, because the high number of states. So, although VQ discretizations provide good solutions, finding the right number of states to use is a hard problem to solve.

Learning with ENNC-QL has been performed with the same parameters and data. The only parameter introduced to the ENNC classification algorithm is the number of cycles, N (defined in section 3.1), set to 2000 for this experiment . However, given that the classifier learned in each iteration uses the classifier obtained in the previous iteration as seed (only small modifications are expected), the number of cycles is reduced in every iteration of the ENNC-QL in 100 cycles, arriving to a minimum value of 200 (big enough to perform the required modifications).

To define the number of iterations of the algorithm to execute (parameter k), it is useful to check Figure 9. The Figure shows the number of prototypes of the state space representation achieved for each action in each iteration. These values are increased very fast up to iteration 34, where the growth stabilizes to the range 1600-1800 in the following 10 iterations. So, the result of iteration 34 is the one used for the rest of the experiment, and consists of 4 state space representations, for actions “Go East”, “Go North”, “Go West” and “Go South” of 1711, 1695, 1587 and 1678 prototypes respectively.

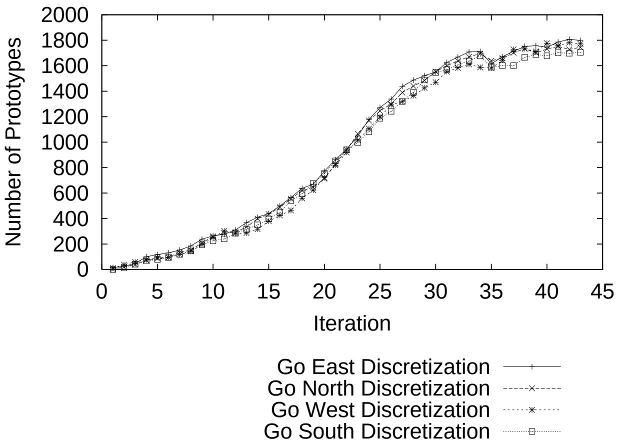


Figure 9. Evolution of the Number of Prototypes defined by the ENNC-QL Algorithm.

Following the steps defined in Figure 3, the non-deterministic Q-Learning update function is used to absorb the non-determinism of the representation, using the same parameters than for VQ and uniform discretizations, but initializing the Q values to the values achieved in the previous step (improving the convergence of Q). After that, the ENNC-QL algorithm achieves a 69.15% of successful trials. The real improvement of the algorithm is not only that it achieves better solutions than uniform and vector quantization discretizations, but that it achieves these solutions without defining the number of discretized states. Figure 10 shows the discretization of the environment for action “Go East”.

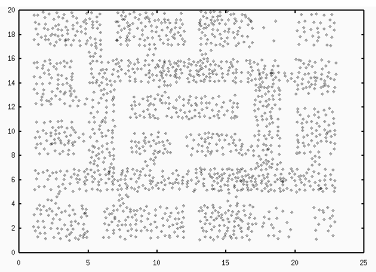


Figure 10. Prototypes using ENNC-QL Discretization for Action “Go East”.

5 CONCLUSIONS AND FURTHER WORK

This paper has shown the relationship between the use of discretization methods of the state space as a generalization technique in reinforcement learning domains and the determinism of these domains. The main conclusion to point out is that even if the original environment is deterministic, the use of these generalization techniques may introduce non-determinism that can result in bad solutions of the problem. We have described how the ENNC-QL algorithm learns the state space representation at the same time than the value function. This allows to supervise the learning of the state space representation in each iteration with the value function learned up to that moment. Thus, the regions are located in the best way so the value function approximation learned in the following iteration of the algorithm is more adjusted to the real one. This allows to achieve better results than other techniques that do not supervise the learning of the state space representation with the value function that is being learned.

Furthermore, the ENNC-QL algorithm automatically computes the appropriate number of regions needed in the new representation, so no knowledge about the size of the domain must be introduced “a priori”. Also, that number of states does not grow much, but it should be preferred to improve this value by forcing the ENNC classifier to generate a reduced set of prototypes in each iteration.

Future work will mainly follow three research lines. First, to eliminate some of the parameters of the algorithm. For instance, the number of iterations to execute, k, can be replaced by an end condition that takes into account the learning convergence. Second, we will compare with other techniques that discretize/generalize the state space in different domains. Lastly, we will verify how the approach can be used in inherent stochastic domains.

ACKNOWLEDGEMENTS

This work has been partially funded by a grant of the Spanish Education Department, and a grant from Spanish Science and Technology Department number TAP1999-0535-C02-02.

References

[1] J.S. Albus, *Brain, Behaviour, and Robotics*, Byte Books, Peterborough, NH, 1981.
[2] R. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ., 1957.
[3] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, Bellmon, Massachusetts, 1996.
[4] Justin A. Boyan and Andrew W. Moore, ‘Generalization in reinforcement learning: Safely approximating the value function’, *Advances in Neural Information Processing Systems*, **7**, (1995).
[5] D. S. Broomhead and D. Lowe, ‘Multivariable functional interpolation and adaptive networks’, *Complex Systems*, (1988).
[6] Richard O. Duda and Peter E. Hart, *Pattern Classification and Scene Analysis*, John Wiley And Sons, 1973.
[7] Fernando Fernández and Daniel Borrajo, ‘VQQL. Applying vector quantization to reinforcement learning’, in *RoboCup-99: Robot Soccer World Cup III*, number 1856 in Lecture Notes in Artificial Intelligence, 292–303, Springer Verlag, (2000).
[8] Fernando Fernández and Pedro Isasi, ‘Designing nearest neighbour classifiers by the evolution of a population of prototypes’, in *Proceedings of the European Symposium on Artificial Neural Networks (ESANN’01)*, pp. 172–180, (2001).
[9] G. E. Hinton, ‘Distributed representations’, Technical report, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA, (1984).
[10] Teuvo Kohonen, *Self-Organizing Maps*, Springer, Berlin, Heidelberg, 1995. (Second Extended Edition 1997).
[11] L.-J Lin, ‘Programming robots using reinforcement learning and teaching’, in *Proceedings of the Nineth National Conference on Artificial Intelligence*, (1991).
[12] S. P. Lloyd, ‘Least squares quantization in *pcm*’. Unpublished Bell Laboratories Technical Note. Portions presented at the Institute of Mathematical Statistics Meeting Atlantic City, New Jersey, September 1957. Published in the March 1982 special issue on quantization of the IEEE Transactions on Information Theory, 1957.
[13] Rémi Munos and Andrew Moore, ‘Variable resolution discretization in optimal control’, *Machine Learning*, (1999).
[14] Juan C. Santarmaría, Richard S. Sutton, and Ashwin Ram, ‘Experiments with reinforcement learning in problems with continuous state and action spaces’, *Adaptive Behavior*, **6**(2), 163–218, (1998).
[15] W. D. Smart and L. P. Kaelbling, ‘Practical reinforcement learning in continuous spaces’, in *Proceedings of the International Conference of Machine Learning*, pp. 903–907, (2000).
[16] C. J. C. H. Watkins, *Learning from Delayed Rewards*, Ph.D. dissertation, King’s College, Cambridge, UK, 1989.