Q-learning for Robots

Claude F. Touzet

Introduction

Robot learning is a challenging – and somewhat unique – research domain. If a robot behavior is defined as a mapping between situations that occurred in the real world and actions to be accomplished, then the supervised learning of a robot behavior requires a set of *representative* examples (situation, desired action). In order to be able to gather such learning base, the human operator must have a deep understanding of the robot-world interaction (i.e., a model). But, there are many application domains where such models cannot be obtained, either because detailed knowledge of the robot's world is unavailable (e.g., spatial or underwater exploration, nuclear or toxic waste management), or because it would be to costly. In this context, the *automatic* synthesis of a representative learning base is an important issue. It can be sought using reinforcement learning techniques – in particular Q-learning which does not require a model of the robot-world interaction. Compared to supervised learning, Q-learning examples are triplets (situation, action, *Q* value), where the *Q* value is the *utility* of executing the action in the situation. The supervised learning base is obtained by recruiting the triplets with the highest utility.

Because it allows the synthesis of behaviors despite the absence of a robot-world interaction model, Q-learning (Watkins 1989) has become the most used learning algorithm for autonomous robotics. Although the convergence theorem does not apply to the robotics domain (due to the limited number of situation-action pairs that can be explored during the life-time of the robot batteries), heuristically adapted Q-learning has proved successful in applications such as obstacle avoidance, wall following, go-to-the-nest, etc. This is mostly due to *neural-based* implementations such as multilayer perceptrons trained with backpropagation, or self-organizing maps. Such implementations provide an efficient generalization, i.e., fast learning, and designate the critic – the reinforcement function definition – as the real issue. The articles REINFORCEMENT LEARNING and REINFORCEMENT LEARNING IN MOTOR CONTROL provide background information on reinforcement learning. Kaelbling (1996) and Sutton (1998) are two other sources of information. For more detailed treatments, the reader should consult Touzet (1997).

Q-learning

Figure 1 shows a functional decomposition of Q-learning. Three different functions are involved: *evaluation*, *memorization* and *updating*. Using the information stored in the robot memory, the current situation is evaluated to select the *best* action to accomplish (i.e., the most reward-promising action). This proposition is modified so as to allow exploration of the situation-action space. The new situation, entered as a consequence of the execution of the action, is qualified by the reinforcement function. Its qualitative criterion (reinforcement) is used by the updating algorithm to adjust the *Q* values in the following way:

$$Q(s,a)_{new} = Q(s,a)_{old} + ß(r + \gamma.Max(Q(s',a)) - Q(s,a)_{old}) \quad (1)$$

where s is the situation, a is the action, r is the reinforcement and s' represents all situations that can be reached from s. ß and γ are positive coefficients less than 1.

Convergence

Recent developments in the theory of reinforcement learning have allowed proof of asymptotic convergence (Dayan 1994). These proofs rely on several assumptions that do *not* apply to robots facing real-world tasks. In particular, the asymptotic convergence requires a discrete coding of the situation-action pairs (look-up table storage) and to try out every action for every situation an infinite number of times. A robot is a mechanical device than needs at least a few hundred microseconds to execute any action. Therefore, due to battery-life that typically last less than 10 hours, only a few thousand situation-action can be visited during a given experiment. This is an extremely small number when compared to the potential number of situation-action pairs (e.g., 10^{26} for the Khepera miniature mobile robot, today the most common research robot). Thus, generalization between similar situation-action pairs is *mandatory*.

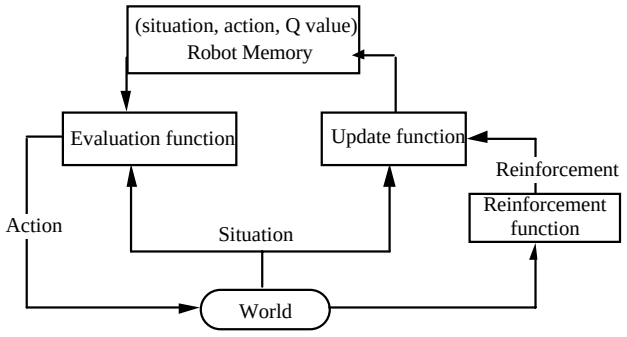

Figure 1. Q-learning method functional decomposition. In response to the present situation, an action is proposed by the robot memory. This action is the one that has the best probability of reward. However, this proposition may be modified by the evaluation function to allow an extensive exploration of the situation-action space. After the execution of the action by the robot in the real world, a reinforcement function provides a reinforcement value. This value – a simple qualitative criterion (e.g., +1, -1 or 0) – is used by the updating algorithm to adjust the utility value associated with the situation-action pair.

Generalization

Improvements emphasizing generalization have been proposed by Mahadevan et al. (1992) who use weighted Hamming distance to generalize between similar situations. This simple method is limited to *syntactic* situation criteria (i.e., it is dependent on the coding of the situations). A second method, proposed by the same authors, adds the action into the syntactic criteria, using clusters to generalize across similar situation-action sets. One of the problems is that the clusters have to be handpicked.

Neural Q-learning

Neural implementations offer a *compact* representation (i.e., limited memory requirement) and good generalization performance (as demonstrated by numerous connectionist applications). The memorization function uses the weight set of the neural network: the memory size required by the system to store the knowledge is defined, *a priori*, by the number of connections in the network. It is independent of the number of explored situation-action pairs. The proposed action is the processing result of the situation by the network, plus the addition of a random component for the exploration. The update function uses the weight modification algorithm to store the utility values computed by the Q-learning rule (1).

The ideal neural implementation would provide, in a given situation, the best action to undertake and its associated *Q* value. However, training of a such network requires the definition of an error on the output layer, i.e., knowledge of the best action to undertake in every situation. Such knowledge can be inferred if there is only two different possible actions for the robot as in the cart pole balancing problem (Barto et al. 1983). However, in the general case, the number of actions is larger. Lin (1993) – who proposed the first multilayer perceptron implementation of the Q-learning (Q-Con) – uses as many perceptrons as there are actions, each network output coding for the utility of accomplishing this action in the current situation. Therefore, only one Q-Con network is updated at every time step, and generalization between networks (i.e., actions) is impossible. Other multilayer perceptron implementations have been proposed (Ackley and Littman 1991, Touzet 1997), but they do not yet solve the output error definition problem.

Q-Kohon

Unsupervised learning models – such as SELF-ORGANIZING FEATURE MAPS: KOHONEN MAPS (q.v.), RADIAL BASIS FUNCTION NETWORKS (q.v.), ADAPTIVE RESONANCE THEORY (ART) (q.v.), CMAC – do not require an error definition for updating their weight values. Q-Kohon, a Kohonen map implementation of the Q-learning, is a method of state grouping involving syntactic similarity and locality (McCallum 1995). Each neuron codes a particular triplet (situation, action, *Q* value); therefore the number of neurons equals the number of stored associations. The neighborhood property of the self-organizing map accounts for the generalization across similar situation-action pairs.

Q-Kohon uses the self-organizing map as an *associative memory*. This associative memory stored triplets. Part of a triplet is used to probe the self-organizing map in search of the corresponding information. Here, situation and *Q* value are used to find the action: the best action to undertake in a world situation is given by the neuron that has the minimal distance to the input situation and to a *Q* value of value +1. The selected neuron corresponds to a triplet (situation, action, *Q*). It is this particular action that should offer the best reward in the world situation. To update the *Q* value, equation (1) requires the maximum Q value of the new entered situation. This is easily obtained by probing the map with the new situation and a *Q* value of value + 1. The selected neuron *Q* value will be the maximum possible value.

A nice side effect of using clustering techniques to implement the Q-learning is that the learned behavior can be interpreted by looking (see SELF-ORGANIZING FEATURE MAPS: KOHONEN MAPS) at the network weights (something extremely difficult with multilayer implementations). Also, because the neurons of the self-organizing map approximate the probability density function of the inputs, one can predict that if a correct behavior is learned, all neurons will code positive *Q* values. This is most useful to determine when a correct behavior has been learned. This last fact results in the *optimization* of the stored knowledge.

	Q-learning	+ Hamming	+ clustering	Q-Comp	Q-Kohon
Time length	55 mn	25 mn	30 mn	8 mn	2 mn
# iterations	7500	3500	4000	2000	500
Memory size	6400	6400	$1.6\ 10^6$	56	176

Table 1. Comparison of various implementations. The learning time is the time in seconds needed to synthesize an obstacle avoidance behavior. It reflects the number of real world experiments required. The number of learning iterations is the number of updates to the memory (look-up table or neural network). The memory size is the number of floats required to store the information.

Comparisons

Experiments aimed at comparing various implementations of the Q-learning in a task of synthesizing an *obstacle avoidance behavior* for the miniature robot Khepera (Touzet 1997) demonstrate that neural Q-learning implementations require alot less memory, less learning examples and learn faster (cf. table 1). The Q-Kohon implementation also exhibits the best behavior after learning, i.e., less negative reinforcements received than all the other implementations.

Reinforcement Function Design

The reinforcement function quality is intrinsically limited by the expert's abilities. When a reinforcement learning experiment does not converge, it is impossible to know if this is due to the fact that the experiment was too short and more examples are needed, or if the intrinsic nature of the reinforcement function forbids convergence. Today, reinforcement learning researchers use a slow-and-painful *trial and error* approach to define the reinforcement function. In the meantime, efforts have been devoted to find ways to automatically define such functions. Santos et al. (1999) have proposed an Update Parameter Algorithm (UPA) to automatically adjust the threshold values: θ_+ and θ_- within a particular definition of the reinforcement function:

$$RF(s_1,...,s_u) = \begin{cases} +1 \; if \; g_1(s_1,...,s_u) > \theta_+ \\ -1 \; if \; g_2(s_1,...,s_u) < \theta_- \\ 0 \quad otherwise \end{cases}$$

where $(s_1,...,s_u)$ is the output readings of the sensors, $g_1(\,)$ and $g_2(\,)$ are any functions linking the sensor data to the rewards.

The resulting effect is to *optimize* the exploration part of the learning phase by achieving and maintaining pre-defined ratios of positive and negative rewards. If there is no positive reward, the evaluation function built during the learning phase will have "0" as maximum value and the policy cannot select effective actions. If there is no negative reward, the robot can remain in a dead-end situation forever. If there is no null reward, the evaluation function will be non-continuous at the frontier between positive and negative situation-action pairs.

A dynamic version of UPA (Santos et al. 1999) updates the threshold values during the learning phase – exploration and exploitation (so as to take into account the improvement of the robot policy). It allows behavior performance improvements without the need of some sort of external supervisor, capable of ranking situations by difficulty and of choosing tasks of increasing difficulty (Dorigo et al. 1998). Santos et al. have been able to synthesize – *for the first time* – a wall following behavior by reinforcement learning (cf. figure 2), a demonstrative support for reinforcement function design techniques.

Discussion

Q-learning is the most used (reinforcement) learning technique for behavior-based robots (see REACTIVE ROBOTIC SYSTEMS). Neural based Q-learning implementations provide compactness and generalization. Clustering-based neural based methods, such as Q-Kohon, allow drastic reduction of the learning time and number of examples required. Their efficiency put forward the reinforcement function definition as one of the major issues.

Another major issue is to be able to overcome the exponentially growing number of required learning examples that comes with target behaviors of greater complexity. Battery-life time seems to impose a definite limit, and researchers tend to promote knowledge incorporation as a speed-up mechanism. The goal is to bias the exploration towards "rewarding" part of the search space – at the expense of *tabula rasa* methods. The drawback is that new – unforeseen – solutions *cannot* be discovered.

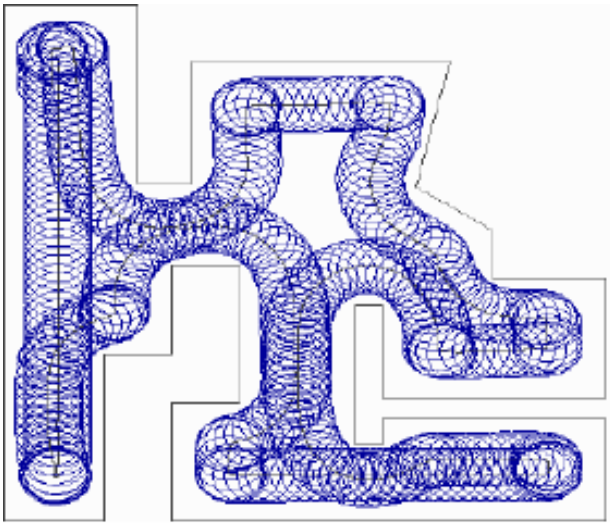

Figure 2. The trace of the miniature Khepera robot after the synthesis of a wall following behavior using a RBF implementation of the Q-learning and Dynamic-UPA in a new environment. Only about 2000 learning iterations are needed.

Lazy learning (Aha 1997), also called instance-based learning, provides a way to add samples without implying bias. In a lazy learning approach, the computation of the inputs is delayed until the necessity arises. Lazy learning samples the situation-action space, storing the succession of events in memory and, when needed, probes the associative memory for the best move. The sampling process stores the successive situation-action pairs generated by a random action selection policy. The exploration phase is done only once, stored and used later by all future experiments. The probing of the memory involves complicated computations: clustering, pattern matching, and so forth.

By storing situation-action pairs, a lazy memory builds an non-explicit model of the situation transition function, that is used as a bias to leverage the model-free following learning phase (i.e., Q-learning). Sheppard et al. (1997) propose to mix lazy learning and reinforcement learning, probing the memory with the reinforcement function. Their objective is to provide a method for predicting the rewards for some state-action pairs without explicitly generating them. They call their algorithm *lazy Q-learning*. For the current real world situation, a situation matcher locates all the states in the memory that are within a given distance. If the situation matcher has failed to find any nearby situations, the action comparator selects an action at random. Otherwise, the action comparator examines the expected rewards associated with each of these situations and selects the action with the highest expected reward. This action is then executed, resulting in a new situation. There is a fixed probability of generating a random action regardless of the outcome of the situation matcher. New situation-action pairs are added to the memory, along with their *Q* values computed in the classical way. Among similar situation-action pairs in the memory, an update of the stored *Q* values is made. There is a limit to the genericity of this lazy memory because the *Q* values associated with the situation-action pairs only apply for a particular application.

Learning is not restricted to single robots. Learning in *cooperative robotics* promises are beguiling: a way to program a set of robots without having to explicitly model their interactions with the world – including the other team members – to achieve cooperation. To achieve this goal, mechanisms that relay the *unique* information associated with the team behavior (reinforcement value) to the individual robots have to be found. Results from the multi-agent research community cannot be applied since they are usually symbolic methods, where robot Q-learning requires a sub-symbolic approach.

Despite all the efforts and success around Q-learning, there are several drawbacks associated with supervised and reinforcement learning when in comes to real applications. First, the time needed to achieve the synthesis of any behavior is *prohibitive*. Second, the robot behavior during the learning phase is – by definition – bad, it may even be *dangerous*. Third, except within the lazy learning approach, a new behavior implies a *new* learning phase. What is needed is a learning that instantaneously synthesizes any behavior, and which performance improves by the mere repetition of this behavior (for further discussion see Touzet (1999)).

References

Ackley D. and M. Littman, 1991. Interactions Between Learning and Evolution, in Artificial Life II, SFI Studies Sc. Complexity, vol.X, (C. G. Langton & Co, eds.) Addison-Wesley, pp. 487-509.

*Aha D. (ed.), 1997. Lazy Learning, Dordrecht:Kluwer Academic Publishers (reprinted from Artificial Intelligence Review, 11:1-5).

Barto A. G., Sutton, R. S. and Anderson, C. W., 1983. Neuron like elements that can solve difficult learning control problems, IEEE Trans. Sys. Man Cybern., 13:835-846.

Dayan P. and T. Sejnowski, 1994. TD(λ) convergences with probability 1, Machine Learning, 14(3): 295-301.

*Dorigo, M. and M. Colombetti, 1998. Robot Shaping: An Experiment in Behavior Engineering, MIT Press.

*Kaelbling L., M. Littman and A. Moore, 1996. Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research 4:237-285.

McCallum, R. A., 1995. Instance-based State Identification for Reinforcement Learning, in Advances In Neural Information Processing Systems 7, MIT Press.

Lin L-J., 1993. Reinforcement Learning for Robots Using Neural Networks, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, CMU-CS-93-103.

Mahadevan S. and J. Connell, 1992. Automatic Programming of Behavior-based Robots using Reinforcement Learning, Artificial Intelligence, vol. 55, Nos. 2-3, pp. 311-365.

Santos J. M. and C. Touzet, 1999. Dynamic Update of the Reinforcement Function during Learning, Connection Science, Special issue on Adaptive Robots, (C. Torras guest ed.), (to appear).

Sheppard J. W. and S. L. Salzberg, 1997. A Teaching Strategy for Memory-Based Control, in Lazy Learning, (D. Aha, ed.), Dordrecht:Kluwer Academic Publishers, pp. 343-370.

*Sutton R. and A. Barto, 1998. Reinforcement Learning, Cambridge, MA: MIT Press.

Touzet C., 1997. Neural Reinforcement Learning for Behaviour Synthesis, Robotics and Autonomous Systems, Special issue on Learning Robot: the New Wave, (N. Sharkey guest ed.), vol. 22, Nos. 3-4, pp 251-281.

Touzet C., 1999. Programming Robots with Associative Memories, in Proc. of International Joint Conf. on Neural Networks, Washington D.C., USA, July 10-16.

Watkins J. C. H., 1989. Learning from Delayed Rewards, Ph.D. thesis, King's College, Cambridge, England.