Reinforcement Learning: an Overview

Pierre Yves Glorennec
INSA de Rennes / IRISA
glorenne@irisa.fr

Abstract

Reinforcement learning relies on the association between a goal and a scalar signal, interpreted as reward or punishment. The objective is not to reproduce some reference signal, but to progessively find, by trial and error, the policy maximizing the rewards. This paper presents the basis of reinforcement learning, and two model-free algorithms, Q-Learning and Fuzzy Q-Learning.

1 Presentation

The objective of this paper is to present the main lines of reinforcement learning (RL), without pretending to make a complete tour of it. A more deepened survey is in preparation. More modestly, the three following points will be highlighted:

- a general presentation of RL, with a theoretical development for a better understanding,
- the description of some algorithms, in order to permit immediate applications,
- the presentation of recent developments on the reciprocal contributions of fuzzy logic and RL.

Contrary to the well-known supervised learning paradigm, that aims to model an input/output mapping, the reinforcement learning approach tries to make emerge *behaviors* permitting to reach an objective, without other information that a scalar signal, the reinforcement. In this type of training, an "agent" [1] is permanently analyzing the consequences of its actions, while having tendency to preferentially replicate those that, in the same circumstances, drove to successes.

One can distinguish two main approaches for the resolution of such a type of problems:

- a research in the space of behaviors, to determine those that permit to achieve the assigned task; this research generally use Genetic Algorithms [15, 35];
- the utilization of Dynamic Programming methods, formalizing this type of training by a Markovian Decision Problem .

These two approaches give in practice comparable results [35], but the second has the advantage to offer a mathematical basis permitting to understand the process better: therefore it is developed here.

The plan of the paper is the next one.

- Section 2 gives basic elements of all RL algorithms.
- The formalism of Markovian Decision Problems, presented in the following section, permits to establish a bridge between the reinforcement learning and methods of Dynamic Programming [4]. This formalism gives a theoretical basis for modeling the interactions between an agent and its environment. This section can be omitted in a first reading.
- The Temporal Differences (TD) method, permitting to evaluate incrementally a policy, is presented in section 4. This method, formalized by Sutton [27], is the basis for most of RL algorithms.

[1] Agent is a very general term that designates here a system under training: a Neural Network, a Fuzzy Inference System, a computer program, etc.

- The Q-Learning, an algorithm based on the Temporal Differences and used to determine an optimal policy, makes the object of section 5. This algorithm is one of those for which the theory is most advanced and for which proofs of convergence exist. It doesn't require the knowledge of probability transitions from a state to another and is model-free.
- Some particular implementation problems are listed in section 6: choice of the reinforcement signal, exploration/exploitation dilemma, representing the Q-values and speeding-up learning methods.
- A "fuzzy" version of Q-Learning is presented in section 7.3. This approach presents several advantages: it permits to treat continuous state and action spaces, to store the state-action values and to introduce *a priori* knowledge.

2 Reinforcement Learning

2.1 General Presentation

Reinforcement learning concerns a family of problems in which an agent evolves while analyzing consequences of its actions, thanks to a simple scalar signal (*the reinforcement*) given out by the environment.

This general definition puts in evidence two important features:

- the agent interacts with its environment and the pair "agent + environment" constitutes a dynamic system;
- the reinforcement signal, that is generally perceived in terms of reward or punishment, permits to the agent to modify its behavior.

In supervised learning, called also "learning with a master", the learning system knows the error that it commits at all times: for each input vector, the corresponding desired output is known. This difference between the actual and the reference output can be used to modify parameters.

In reinforcement learning, or "learning with a critic", the received signal is the sanction (positive, negative or neutral) of a behavior: this signal indicates *what* you have to do without saying *how* to do it. The agent uses this signal to determine a policy permitting to reach a long-term objective.

Another difference between these two approaches is that reinforcement learning is fundamentally online, because the agent's actions modifies the environment: to accomplish its task, the agent must link several actions, *i.e.* follow a *policy* and, more precisely, to determine the policy that will maximize the future rewards.

The general process, schematized in Figure 1, is the following :

1 - at time step t, the agent is in state $x(t)$,

2 - it chooses one of the possible actions in this state, $a(t)$,

3 - it applies the action, what provokes,:

- the passage to a new state, $x(t+1)$,
- the receipt of the reinforcement, $r(t)$;

4 - $t \leftarrow t+1$

5 - go to 2 or stop if the new state is a terminal one.

Let $\mathcal{X}$ be the set of states and $\mathcal{A}$ the set of actions. The reinforcement $r(t)$ is the consequence of action $a(t)$ chosen in state $x(t)$. The reinforcement function is an application of the product space $\mathcal{X} \times \mathcal{A}$ in $\mathcal{R}$ ($r : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{R}$). In a first time, spaces $\mathcal{X}$ and $\mathcal{A}$ are supposed *discrete*. Extensions to continuous state and action spaces will be treated in paragraphs 6.3 and 7.3.

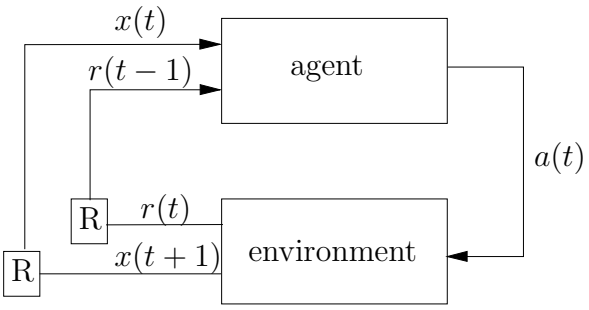

Figure 1: Modeling of interactions between the agent and its environment (R represents a delay)

2.2 The return function

The agent's goal is to maximize the accumulated future rewards. The return function, or simply the return, $R(t)$, is a long-term measure of rewards. One finds three possible expressions for the return. In case of stochastic reinforcement, we have to take the expectation of the sum in the following equations.

The finite-horizon model
In this case, the horizon corresponds to a finite number of steps in the future. It exists a terminal state and the sequence of actions between the initial state and the terminal one is called a *period*. One has:

$$R(t) = r_t + r_{t+1} + \ldots + r_{t+K-1} \tag{1}$$

where K is the number of steps before the terminal state. This number K can be fixed *a priori* or not.

The discounted return (infinite-horizon model)
One uses this criteria when the sequence of actions is infinite. A *discount factor*, γ, $0 \leq \gamma \leq 1$, is introduced and the criteria becomes:

$$R(t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{2}$$

The value of γ permits to modulate the period the agent takes the reinforcements into account. If $\gamma = 0$, the agent is "myopic" and considers the immediate reward only; the more γ is near of 1, the more the agent looks ahead.

The average-reward model
An algorithm of reinforcement learning, the R-Learning [11, 26], uses a third criteria, that takes the average of future reinforcements:

$$R(t) = \lim_{n \to \infty} \frac{1}{n} \sum_{k=0}^{n} r_{t+k} \tag{3}$$

In the following, we only use the discounted return, which is also used in the Dynamic Programming approach.

3 Mathematical Framework

The interaction between the agent and its environment can be modeled as a *Markovian Decision Problem*, that one can solve by Dynamic Programming methods [4]. One considers a stochastic process, X, whose possible values belong to a finite number of states, $\mathcal{X} = \{x_0, x_1, \ldots, x_K\}$.

In a Markovian Decision Problem, transitions from state i to state j depend only on the allowed actions in state i. Let $\mathcal{A}_i$ be the set of possible actions in state i. Probabilities of transition are:

$$p_{ij}(a) = Pr(X_{t+1} = x_j | X_t = x_i, a(t) = a) \tag{4}$$

where $a \in \mathcal{A}_i$.

The objective of a Markovian decision problem is, for every initial state, to find an *optimal policy*, that is a sequence of actions that maximizes the return.

3.1 The value function

In every state, x_t, an agent chooses an action according to a certain policy, π. One notes:

$$a_t = \pi(x_t) \tag{5}$$

One values a policy by the expectation of future reinforcement with the discounted return. The value of state x under the policy π is given by:

$$V^{\pi}(x) = E\{\sum_{t=0}^{\infty} \gamma^t r(x_t, \pi(x_t)) | x_0 = x\} \ \forall x \in \mathcal{X} \tag{6}$$

where $E\{r(x_t, \pi(x_t))\}$ represents the expected reinforcement when one applies the action proposed by the policy π in the state x_t. To simplify, we write $R(x, a) = E\{r(x, a)\}$. The function $V^{\pi}(x)$ can be recursively rewritten:

$$V^{\pi}(x) = R(x, \pi(x)) + \gamma \sum_{y \in \mathcal{X}} p_{xy}(\pi(x)) V^{\pi}(y) \tag{7}$$

where $p_{xy}(\pi(x))$ is the probability to pass from state x to y while applying the action $\pi(x)$.

The function $x \rightarrow V^{\pi}(x)$ defines the value function under the policy π. One can calculate it off-line if one knows the initial state and all probability transitions, that is to say if one knows a model. The problem is then about solving the linear system of equations given by equation 7. One also knows that it exists an optimal policy, noted $\pi^{\star}$ and defined by:

$$V^{\pi^{\star}}(x) \geq V^{\pi}(x), \ \forall x \in \mathcal{X}, \ \forall \pi \tag{8}$$

To simplify notations, we write $V^{\star} = V^{\pi^{\star}}$. The optimal value of a state is:

$$V^{\star}(x) = \max_{\pi} E\{\sum_{t=0}^{\infty} \gamma^t r_t\} \tag{9}$$

One shows that $V^{\star}$ satisfies, in the case of an infinite horizon, the following equation, known as Bellman optimality equation (or equation of Dynamic Programming):

$$V^{\star}(x) = \max_{a \in \mathcal{A}_x} \{R(x, a) + \gamma \sum_{y} P_{xy}(a) V^{\star}(y)\}, \ \forall x \in \mathcal{X} \tag{10}$$

The optimal policy is deducted of $V^{\star}$ by:

$$\pi^{\star}(x) = \arg \max_{a \in \mathcal{A}_x} \{R(x, a) + \gamma \sum_{y} p_{xy}(a) V^{\star}(y)\} \tag{11}$$

3.2 Solving Bellman equation

It exists two usual methods to calculate the optimal policy,

- *policy iteration*, which manipulates policies directly,
- *value iteration*, looking for the optimal value function .

These two methods give specific RL algorithms.

3.2.1 Policy iteration

The idea consists in starting from any initial policy and improving it iteratively. The algorithm is the next one :

Policy Iteration Algorithm

choose an initial policy , $\hat{\pi}$
repeat
  $\pi = \hat{\pi}$
  calculate V^{π} (equation 7)
  improve the policy for every state:
    $\hat{\pi}(x) = \arg\max_a \{R(x,a) + \gamma \sum_y p_{xy}(a) V^{\pi}(y)\}$
until the policy remains unchanged

The value of the policy is the solution of a linear system of equations. Then, for every state, one watches if it is possible to improve the policy while changing only the first action.

One notes $|\mathcal{E}|$ the cardinal of a set $\mathcal{E}$. There is therefore $|\mathcal{A}|^{|\mathcal{X}|}$ possible policies. The iteration on policies is fast when the action space is small and sometimes some iterations are sufficient; on the other hand this method is very expensive, or even intractable, in case of large action space.

3.2.2 Value Iteration

In this method, one doesn't try to determine the policy explicitly, but the action of optimal value for every state. One defines the value of an action, $Q^{\pi}(x,a)$, as the discounted return if action a is applied in state x and if the policy π is followed then:

$$Q^{\pi}(x,a) = R(x,a) + \gamma \sum_y p_{xy}(a) V^{\pi}(y) \tag{12}$$

The function $Q^{\star} = Q^{\pi^{\star}}$ corresponds to the optimal policy and the Bellman optimality equation becomes:

$$V^{\star}(x) = \max_{a \in \mathcal{A}_x} Q^{\star}(x,a) \tag{13}$$

To evaluate $V^{\star}(.)$, one supposes that the horizon is finite, considering firstly one stage, then two and so forth. Let $V_n(.)$ be the evaluation of the value function, found after n iterations. The algorithm is as follows:

Value Iteration Algorithm

choose an initial value function , $V_0(x)$, for all x of $\mathcal{X}$
repeat for $n \geq 0$
  for all x of $\mathcal{X}$
    calculate $Q_{n+1}(x,a) = R(x,a) + \gamma \sum_y p_{xy}(a) V_n(y)$, for all a of $\mathcal{A}$
    $V_{n+1}(x) = \max_{a \in \mathcal{A}_x} Q_{n+1}(x,a)$
until a stopping condition

One shows that this algorithm converges in a finite number of iterations toward the optimal policy. More precisely, one has:

$$(\forall \epsilon)(\exists M)(\forall x \in \mathcal{X})(||V_M(x) - V^{\star}(x)|| \leq \epsilon) \tag{14}$$

The architecture of algorithms based on value iteration of is presented in Figure 2.

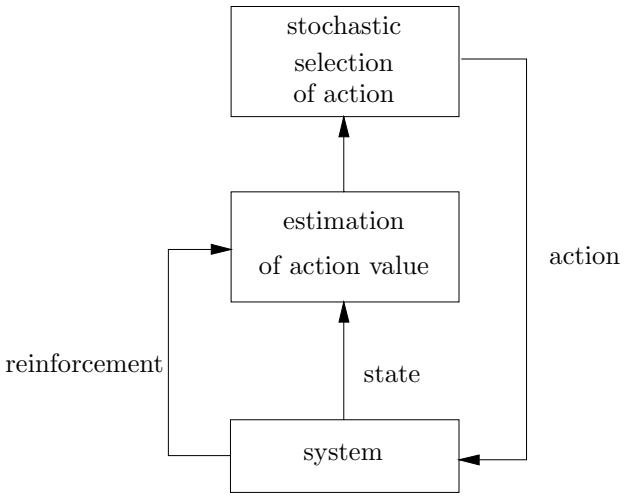

Figure 2: Architecture for value iteration

4 Temporal Differences

4.1 TD(0)

We consider the case where the agent doesn't know its environment (nor the expectation of immediate reinforcement, nor probabilities of transition from a state to another according to the chosen action). It is possible to build a model and to use it to learn the value function: one speaks then of indirect methods. One can also estimate directly the value function from experience: it is the case of methods without models, methods that will be developed exclusively in the following.

As the value of a state is not known, the idea of Sutton, formalized in the Temporal Difference method [27], is to use the successive evaluations of $V^\star$.

At time step t, $\hat{V}_t(x_t)$ represents the evaluation of $V^\star(x_t)$: the action a_t makes pass x_t to x_{t+1} and the immediate reinforcement is r_t. It immediately follows (see equation 7) that a new evaluation of the value becomes:

$$r_t + \gamma \hat{V}_t(x_{t+1}) \tag{15}$$

because all transitions of probability toward another state that x_{t+1} are zero and $\hat{V}_t(x_{t+1})$ represents the evaluation, made at time t, of the value of the state x_{t+1}.

The temporal difference between the two successive evaluations of the value of x_t is $r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)$. This difference is used to update the value:

$$\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \beta\{r_t + \gamma \hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)\} \tag{16}$$

If the learning gain β, $0 < \beta \leq 1$, decreases sufficiently slowly toward zero, the successive evaluations of the value converge toward the optimal value function [27].

4.2 The eligibility trace

The previous algorithm, called TD(0), takes only into account the new state. Besides, only the current state is concerned. Sutton spreads the evaluation for all states, according to their *eligibility*, that is, the degree of visit of a state in the recent past. The eligibility (or the trace of eligibility) can be defined with several ways. The *accumulative* eligibility is defined by:

$$e_t(x) = \begin{cases} 1 + \gamma\lambda e_{t-1}(x) & \text{if } x = x_t \\ \gamma\lambda e_{t-1}(x) & \text{otherwise} \end{cases} \tag{17}$$

while a *replacing* eligibility corresponds to:

$$e_t(x) = \begin{cases} 1 & \text{if } x = x_t \\ \gamma\lambda e_{t-1}(x) & \text{otherwise} \end{cases} \tag{18}$$

The trace of eligibility constitutes a short-term memory of the frequency of visit of a state. It decreases exponentially in the time, unless to be reactivated by a new visit.

While introducing the trace of eligibility, the update of $\hat{V}_{t+1}(x_t)$ becomes:

$$\hat{V}_{t+1}(x_t) = \hat{V}_t(x_t) + \beta\{r_t + \gamma\hat{V}_t(x_{t+1}) - \hat{V}_t(x_t)\}e_t(x) \tag{19}$$

A new parameter, λ, $0 \leq \lambda \leq 1$ is introduced and the corresponding algorithm family is called TD(λ). The best performances are when λ decreases from values near of 1 toward 0, see reference [16].

4.3 TD(λ)

The temporal difference method is used by all algorithms of RL without model, sometimes with $\lambda = 0$, sometimes with $\lambda > 0$ because in a lot of cases the convergence is faster. The complete algorithm is presented below.

Algorithm TD(λ)

$t = 0,\ V_t(x) = 0,\ e_t(x) = 0$ for all x of $\mathcal{X}$
`repeat`
`observe the transition` $x_t \rightarrow x_{t+1}$
`calculate` $e_t(x)$, equation 17 or 18
`to calculate` $\hat{V}(x)$ for all x, equation 19
$t \leftarrow t + 1$
`until a stopping condition`

5 Presentation of Q-Learning

It exists several approaches for reinforcement learning without models. Some are based on policy iteration (see section 3), such as the *Actor Critic Learning*, and others on value iteration, such as Q-Learning or SARSA. The Q-Learning, proposed by Watkins [32], is perhaps the more popular of AR algorithms, by reason of its simplicity. It is the algorithm that is developed therefore in this paper.

5.1 One-step Q-Learning

The first version of Q-Learning is based on the temporal differences of order 0, TD(0), while only considering the following step (*one-step Q-Learning*). The agent observes the present state, x_t, and executes an action, a_t, according to the evaluation of the return that it makes at this stage. It updates its evaluation of the value of the action while taking in account, a) the immediate reinforcement, r_t, and b) the estimated value of the new state, $V_t(x_{t+1})$, that is defined by:

$$V_t(x_{t+1}) = \max_{b \in \mathcal{A}_{t+1}} Q(x_{t+1}, b) \tag{20}$$

The update corresponds to the equation:

$$Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \beta\{r_t + \gamma V_t(x_{t+1}) - Q(x_t, a_t)\} \tag{21}$$

β is a learning rate such that $\beta \rightarrow 0$ as $t \rightarrow \infty$. This equation can be written:

$$Q(x_t, a_t) \leftarrow (1 - \beta)Q(x_t, a_t) + \beta\{r_t + \gamma V_t(x_{t+1})\} \tag{22}$$

This update corresponds to the barycenter of the old and new evaluations, weighted by β.

Q-Learning, in addition to its simplicity, presents several interesting characteristics.

- The evaluations of Q, the Q-values, are independent of the policy followed by the agent. This one can follow any policy, while continuing to construct correct evaluations of the value of actions.

- Q-values are exploitable a long time before the formal convergence that can be sometimes very slow.
- Lastly, there are proofs of convergence toward the optimal policy [32, 33, 31].

But then, the too fast choice of the action having the biggest Q-value

$$a = \arg\max_{b \in \mathcal{A}_x} Q(x, b) \tag{23}$$

can drive to local minima: it is necessary to insure that all actions were tested sufficiently. It is what makes the object of section 6.2.

Remark: a reinforcement different from 0 can sometimes mean the end of a period, for example a mobile robot can receive the reinforcement $r = -1$ when it enters in collision with an obstacle. In this case, there is not a following state and the agent restarts a new sequence of training. The updating equation is then:

$$Q(x_t, a_t) \leftarrow (1 - \beta) Q(x_t, a_t) + \beta r_t \tag{24}$$

5.2 Q(λ)

Q(λ) [23] is a generalization of Q-Learning that uses traces of eligibility: the one-step Q-learning is a particular case with $\lambda = 0$. Traces of eligibility must be defined on the space product $\mathcal{X} \times \mathcal{A}$. Definitions are therefore slightly modified. For example, for the accumulative eligibility, one has:

$$e_t(x, a) = \begin{cases} 1 + \gamma\lambda e_{t-1}(x, a) & \text{if } x = x_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(x, a) & \text{otherwise} \end{cases} \tag{25}$$

and for the replacing eligibility:

$$e_t(x, a) = \begin{cases} 1 & \text{if } x = x_t \text{ and } a = a_t \\ \gamma\lambda e_{t-1}(x, a) & \text{otherwise} \end{cases} \tag{26}$$

The convergence of Q(λ) toward $Q^\star$ is not assured anymore for $\lambda > 0$, but experiences show that the training is faster. The algorithm is summarized below.

Algorithm of Q(λ)

$t = 0$, $Q(x, a) = 0$, $e_t(x, a) = 0$ for all (x, a) of $\mathcal{X} \times \mathcal{A}$
`observe` x_t
`repeat`
  `choose` a_t
  `observe the transition` $x_t \rightarrow x_{t+1}$
  $\epsilon_t^\dagger = r_t + \gamma V_t(x_{t+1}) - Q(x_t, a_t)$
  $\epsilon_t = r_t + \gamma V_t(x_{t+1}) - V_t(x_t)$
  `calculate` $e_t(x, a)$, equation 25 or 26
  `calculate` $Q(x, a)$ for all (x, a) of $\mathcal{X} \times \mathcal{A}$:
    $Q(x_t, a_t) \leftarrow Q(x_t, a_t) + \beta \epsilon_t^\dagger e_t(x_t, a_t)$
    $Q(x, a) \leftarrow Q(x, a) + \beta \epsilon_t e_t(x, a)$ for all pairs different from (x_t, a_t)
  $t \leftarrow t + 1$
`until a stopping condition`

6 Implementation of Q-Learning

As we have seen in the previous section, Q-learning is simple and attractive. However, for an efficient implementation, we have to pay attention to the following problems:

- choosing the reinforcement signal,
- choosing an action in a state,
- representing the Q-values,
- speeding-up learning.

6.1 Reward/punishment functions

It exists several types of reinforcement, that one can classify into two big families.

Shortest path problems
In this case, the reinforcement is equal to -1 in all states different of the desired state, in order to force the agent to reach this state as quickly as possible.

Avoiding problems
The reinforcement is zero in all states, except in case of failure, where it takes the value -1. It is the case, for example, of the inverted pendulum, where a failure occurs when the pole falls or the cart hits the end of the track.

In fact, for one same problem, one can define different reinforcement signals. Some remarks.

- Rewards or punishments can be received after a sequence of actions: one speaks of *delayed* reinforcement. The problem is to identify the contribution of every action to this signal. This important problem is known as *temporal credit assignment problem*.
- The reinforcement signal is often discrete, but it can be also continuous. For example, if the agent's objective is to reach a target, it is not indifferent to know if the target has been missed of some centimeters or about hundred meters: one can conceive a gradual signal.
- The "true" reinforcement signal concerns the goal. However, in certain cases, it can be useful to give some intermediate indications on the achievement of the task (see also section 6.4.2). The difficulty is double:
  - the agent must not lose of view the final objective[2];
  - the agent can be confused when this intermediate signal disappears, problem identified by Whitehead [34] and known as *extinction effect*.
- In spite of its very intuitive aspect, it exists situations where the function of reinforcement is difficult to design [30].
- The nature of reinforcement induces a behavior. If a team of agents receive the same reinforcement signal, heterogeneous behavior can emerge. Balch [1] reports that, in a robot soccer game, a behavioral diversity is encouraged with the following reinforcement:

$$r_t = \begin{cases} 1 & \text{if the team scores} \\ -1 & \text{if the opponent scores} \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

6.2 Exploration/Exploitation

After convergence of Q-Learning, the optimal policy is performed while choosing the action that, to every state, maximize the Q-function:

$$a = \arg\max_{b \in \mathcal{A}_x} Q^\star(x, b) \tag{28}$$

This policy is called *greedy*. However, in the beginning of the training, the values of $Q(x, a)$are not meaningful: applying the greedy policy too quickly often drive to local minima. To get an useful evaluation of Q, it is necessary to sweep and to evaluate the set of possible actions, for all states: it is what one calls

[2]For example, in a game of chess, if the agent is rewarded to every hold of piece, he can tempt to maximize holds to the risk to lose the part.

the phase of *exploration* in opposition to the *exploitation* one, at the end of training.

The Exploration/Exploitation dilemma can be expressed by: at each state, the agent must choose between

- an action for which the expected reward is supposed to be good quality
- or an action whose quality can be, to this precise instant of the choice, less good but whose application could drive it in promising but not explored zones.

The answer to this dilemma is to explore sufficiently and to test all possible actions in all situations, what can take a lot of times. It is necessary to find a compromise therefore and an abundant literature is dedicated to this problem [9, 19, 29, 36].

Three methods of exploration are described below. In the following, the chosen action according to an exploration/exploitation policy will be noted $x \rightarrow a = EEP(x)$.

6.2.1 Pseudo-stochastic Method

- The action with the best value has a probability P to be chosen,
- otherwise, an action is chosen randomly among all possible actions in the given state.

This method is also known as P-greedy method.

6.2.2 Pseudo-exhaustive Method

- The action with the best value has a probability P to be chosen,
- otherwise, one takes the action the least lately chosen in the given state.

6.2.3 Boltzmann Distribution

The action a is chosen with the probability :

$$P(a|x) = \frac{\exp(\frac{1}{T}Q(x,a))}{\sum_b \exp(\frac{1}{T}Q(x,b))} \tag{29}$$

T is comparable to "temperature" in simulated annealing. This parameter decreases in the time.

The pseudo-stochastic and pseudo-exhaustive methods generally give better results that an exploration using the Boltzmann distribution. Caironi showed experimentally that the pseudo-exhaustive method was often better, except in cases where the agent cannot discern between two states (phenomenon known as *perceptual aliasing*) [9].

6.3 Neural Implementation

For discrete state and action spaces of reduced size, Q-values are stocked in a look-up table: $q_{ij} = Q(x_i, a_j)$, for $x_i \in \mathcal{X}$ and $a_i \in \mathcal{A}$. If the size of $\mathcal{X}$ or of $\mathcal{A}$ increases considerably, or if $\mathcal{X}$ is continuous, it becomes impossible to visit all the states and to test all actions in one reasonable time. For this reason, capacities of interpolation of Artificial Neural Networks (ANN) have been exploited [17, 25] in the case of a continuous state space and a discrete action space.

- Let n be the dimension of $\mathcal{X}$. A state $\mathbf{x}$ is a vector[3] of components $x_1, \ldots, x_n$.
- Let J be the number of actions in $\mathcal{A}$. Two neural implementations are possible:
  - one ANN with n inputs and J outputs, every output representing, after training, the Q-value $Q(., a_j)$, $j = 1$ to J,
  - J ANN with one output: one output by action, in accordance with the architecture of Figure 3.

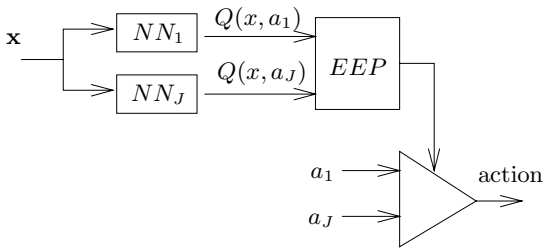

Figure 3: Neural System with J actions

This second architecture, proposed initially by Lin [17], is more effective.

The process is the next one for Q(0):

1. Presentation of the state vector $\mathbf{x}$. Every ANN calculates an evaluation of $Q(\mathbf{x}, a_j)$, $j = 1$ to J.
2. A module of exploration/exploitation, EEP, chooses the action to apply, $a_{j^\star}$.
3. The new state is $\mathbf{y}$ and the immediate reinforcement is r.
4. Presenting $\mathbf{y}$ as a new input of the ANNs, its value is calculated by:

$$V(\mathbf{y}) = \max_{1 \leq j \leq J} ANN_j(\mathbf{y}) \tag{30}$$

   where $ANN_j(\mathbf{y})$ is the output of the ANN number j
5. The new evaluation of $Q(\mathbf{x}, a_{j^\star})$ becomes $r + \gamma V(\mathbf{y})$: the difference between the elder and the new value is considered as the error committed by $ANN_{j^\star}$. This error is used to modify the weights[4].

6.4 Speed up of training

Several authors proposed heuristics aiming to accelerate the training, mainly when the reinforcement is delayed. Among these, one finds the reuse of experience [17], or the training with an external critic [34].

6.4.1 Experience replay

The basic idea of experience replay is to store into a stack the pairs (x, a) and to restart a sequence backward, once received the delayed reinforcement: the recently acquired informations will serve to re-adjust the Q-values.

Consider for example the sequence of actions described in Figure 4. With the action a_1, the system passed from the state x_1 to x_2 and received a reinforcement $r_1 = 0$; then the action a_2 led in x_3 with $r_2 = 0$; lastly the action a_3 drove to the state x_4 with $r_3 = 1$.

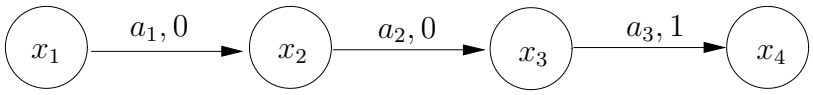

Figure 4: Sequence of actions

While analyzing the algorithm of experience replay, one gets the following informations starting initial condition $Q(x, a) = 0$ for all (x, a) of $\mathcal{X} \times \mathcal{A}$:

1. When the signal $r_3 = 1$ arrives, the stack contains (x_1, a_1) and (x_2, a_2); and we have $Q(x_1, a_1) = Q(x_2, a_2) = 0$, $Q(x_3, a_3) = \beta$ according to equation 24.

[3] Vectors are written in bold type to avoid all confusion with their components.

[4] Reinforcement Learning being sometimes long, it is not judicious to use a slow algorithm as Back-propagation to optimize the ANN. It is better to use a second-order algorithm: *e.g.* quasi-newton or Levenberg-Marquardt algorithms.

2. The extraction from the stack gives successively:
   - $V(x_3) = \beta$ (since only $Q(x_3, a_3)$ is different of 0).
   - After extraction of (x_2, a_2), $Q(x_2, a_2)$ is revalued and takes the value $\beta^2\gamma$.
   - in the same way, after the extraction of (x_1, a_1), one gets $Q(x_1, a_1) = \beta^3\gamma^2$.

The update of Q-values is only done after having received the delayed reinforcement. It is more efficient, because the evaluation of the visited states is more precise.

Algorithm of experience replay

$t = 0$, $Q(x, a) = 0$ for all (x, a) of $\mathcal{X} \times \mathcal{A}$
`empty the stack`
`observe` x_t
`repeat`
  `choose` a_t according to an EEP
  `observe the transition` $x_t \rightarrow x_{t+1}$ and receive r_t
  `if` $r_t = 0$ `then` add (x_t, a_t) to the stack
    `otherwise` $Q(x_t, a_t) \leftarrow (1 - \beta)Q(x_t, a_t) + \beta r_t$
      $k \leftarrow$top of stack
      $x_{k+1} = x_t$
      `while` stack not empty `do`
        `extract` (x_k, a_k)
        $V(x_{k+1}) = \max_a Q(x_{k+1}, a)$
        $Q(x_k, a_k) \leftarrow (1 - \beta)Q(x_t, a_t) + \beta V(x_{k+1})$
        $k = k - 1$
      `done`
`until a stopping condition`

6.4.2 Learning with an external critic

The reinforcement signal fundamentally translates the success or the failure of a mission and, for this reason, it is often delayed. It is sometimes interesting to guide the agent before the arrival of the delayed reinforcement, while rewarding actions "that go in the common sense" or while punishing those that could be dangerous[5].

With this point of view, Whitehead proposed to influence the choice of an action by the EEP by adding to Q-values a bias $B : \mathcal{X} \times \mathcal{A} \rightarrow \mathcal{R}$, defined by:

$$B(x_t, a_t) \leftarrow (1 - \delta)B(x_t, a_t) + \delta r_c(t) \tag{31}$$

where $r_c(t)$ represents the immediate reinforcement given by an external critic when the action a_t has been applied in the state x_t. The EEP is modified therefore by this bias. In the case of a pseudo-stochastic exploration (section 6.2.1), one has:

- $a_t = \arg\max_a\{Q(x_t, a) + B(x_t, a)\}$with a probability P,
- otherwise a_t is an any action in $\mathcal{A}_{a_t}$.

This method remains however sensitive to the ratio between the reinforcement of the critic and the delayed reinforcement, ratio difficile to value *a priori*.

Caironi [9] proposed several improvements to the Q-Learning, combining the reuse of the experience and the training with an external critic. His algorithms are robust and effective.

[5]For example, it is legitimate to punish a mobile robot that would increase its speed whereas conditions of circulation would ask unlike to reduce it.

7 Fuzzy Logic and Reinforcement Learning

Reinforcement learning has been used for optimization of Fuzzy Inference Systems (FIS), with several types of implementation:

- Some of them are based on policy iteration, driving to Actor-Critic architectures, [6, 16],
- the others, based on value iteration, generalize Q-Learning, [12, 22].

In the case of continuous action spaces, the FIS can serve both to the representation of Q-values and the calculation of the action to be applied.

The utilization of a FIS brings several advantages:

- the representation of the Q-values, what is possible thanks to the universal approximation property of most of the FIS;
- the possibility to treat continuous state and action spaces;
- the integration of *a priori* knowledge and the interpretation of the acquired knowledge after the training phase.

7.1 The used FIS

We consider only order-0 Takagi-Sugeno FIS, that are very used for modeling and/or control real systems. Let's recall that such a FIS is described by a set of fuzzy rules such as:

$$\text{if } x_1 \text{ is } A_1^i \text{ and } \ldots \text{ and } x_n \text{ is } A_n^i \text{ then } y = c_i$$

where A_j^i is a fuzzy set on the domain of x_j and c_i is a real number[6], for $j = 1$ to n and $i = 1$ to N, n being the dimension of the input space and N the number of rules.

To make easier the interpretation, we are going to impose the following constraints:

- the structure is fixed *a priori* by the user;
- the partitions on every input domain are triangular strong fuzzy partitions, defined by:

$$(\forall \mathbf{x})(\sum_i \alpha_i(\mathbf{x}) = 1) \tag{32}$$

  where α_i is the truth value of rule i;
- the conjunction in the premise part of the rules is the product.

These constraints are not restrictive but allow simplifications (see eq. 33).
Let $\mathbf{x} = (x_1, \ldots, x_n)^t$ be an input vector. One notes indifferently $\mathbf{x} \rightarrow y(\mathbf{x})$ or $\mathbf{x} \rightarrow FIS(\mathbf{x})$ the input/output mapping. The output of an order-0 Takagi-Sugeno FIS is achieved in three steps by:

1. calculation of membership degrees to the different fuzzy sets, $\mu_{A_j^i}(x_j)$, for $i = 1$ to N and $j = 1$ to n;
2. calculation of the truth value of each rule, given the input vector: $\alpha_i(\mathbf{x}) = \prod_j \mu_{A_j^i}(x_j)$;
3. calculation of the output,

$$y(\mathbf{x}) = \frac{\sum_i \alpha_i(\mathbf{x}) c_i}{\sum_i \alpha_i(\mathbf{x})} = \sum_i \alpha_i(\mathbf{x}) \times c_i \quad \text{(see eq. 32)} \tag{33}$$

[6] In the general case, the conclusion is a function of inputs.

7.2 Representation of the Q-function by a FIS

In a first time, we consider only one action, a. We want to calculate the associated Q-value by the function:

$$\mathbf{x} \rightarrow y = \hat{Q}(\mathbf{x}, a) = FIS(\mathbf{x}),\ \mathbf{x} \in \mathcal{X} \tag{34}$$

As the chosen representation is a fuzzy system, this function is defined by the rules:

$$\begin{array}{ll} \text{if } \mathbf{x} \text{ is } S_1 & \text{then } y = c_1 \\ \ldots & \\ \text{if } \mathbf{x} \text{ is } S_N & \text{then } y = c_N \end{array}$$

where S_i is the premise part of rule i: x_1 is A_1^i and ... and x_n is A_n^i. By abuse of language, S_i will be called also in the following the modal vector[7] correspondent to rule i, or a *prototype* for rule i. Moreover one has: $\alpha_i(S_i) = 1$ and $\alpha_j(S_i) = 0$ for $j \neq i$.

For a given input vector, $\mathbf{x}_t$, the evaluation of the value of action a is given by equation 33:

$$\hat{Q}(\mathbf{x}_t, a) = FIS(\mathbf{x}_t) = \sum_i \alpha_i(\mathbf{x}_t) \times c_i \tag{35}$$

Putting $c_i = q[S_i, a]$, the previous equation becomes:

$$\hat{Q}(\mathbf{x}_t, a) = \sum_i \alpha_i(\mathbf{x}_t) \times q[S_i, a] \tag{36}$$

Proposition 7.1. q-value of an action
In the representation of the function Q by a FIS, the conclusions of rules, $(c_i)_{i=1}^N$, can be interpreted as the restriction of the function $\mathbf{x} \rightarrow \hat{Q}(\mathbf{x}, a)$ to the modal vectors $(S_i)_{i=1}^N$, and one has:

$$q[S_i, a] \equiv \hat{Q}(\mathbf{x}, a), \quad \textit{if } \mathbf{x} = S_i, \textit{ for } i = 1 \textit{ to } N \tag{37}$$

$q[S_i, a]$ is called the q-value of action a in the state S_i (or for the rule i). The representation of $\hat{Q}(., a)$ by a FIS is equivalent to determine the q-values for each of the modal vectors (i.e. for each of rules), then to interpolate for an any input vector.

The fuzzy formalism is an elegant way to pass from a discrete state space (represented by the N modal vectors) to a continuous one.

7.3 Continuous state and action spaces

In the general case, we can use a FIS for both computing the action and its associated Q-value. An elementary rule is written, for $i = 1$ to N:

$$\textbf{If } \mathbf{x} \text{ is } S_i \textbf{ then } y = a_i \textbf{ with } q_i$$

Using equation 33 for an input vector $\mathbf{x}$, one can define the inferred action, $A(\mathbf{x})$, and its associated quality, $\hat{Q}(\mathbf{x}, A(\mathbf{x}))$, by:

$$A(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i(\mathbf{x}) \times a_i \tag{38}$$

$$\hat{Q}(\mathbf{x}, A(\mathbf{x})) = \sum_{i=1}^{N} \alpha_i(\mathbf{x}) \times q_i \tag{39}$$

We emphasize that $A(\mathbf{x})$ is not an explicit input for $\hat{Q}(\mathbf{x}, A(\mathbf{x}))$. Figure 5 shows that there is not any ambiguousness.

[7] A modal vector, in the case of a triangular strong fuzzy partition, has for components the vertices of the triangles in the premise part of a rule.

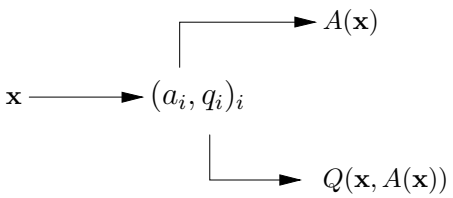

Figure 5: Relations between $\mathbf{x}$, $A(\mathbf{x})$ and $\hat{Q}(\mathbf{x}, A(\mathbf{x}))$.

7.4 Knowledge Extraction

Knowledge extraction is the most natural extension of discrete Q-Learning: as one associates several actions in every state, one associates several competing conclusions to every rule. To each conclusion is associated a q-value that is incrementally updated. The learning process consists then in determining the best set of rules, the one that will optimize the future reinforcements. The initial rule base is composed therefore of N rules such as:

$$\begin{array}{lll} \textbf{If } \mathbf{x} \text{ is } S_i \textbf{ then} & & y = a[i,1] \textbf{ with } q[i,1] = 0 \\ & \textbf{or} & y = a[i,2] \textbf{ with } q[i,2] = 0 \\ & & ..... \\ & \textbf{or} & y = a[i,J] \textbf{ with } q[i,J] = 0 \end{array}$$

where $(a[i,j])_{j=1}^{J}$ are potential solutions whose quality is initialized to zero.

For every rule i, let $k[i] \in \{1, J\}$ be the subscript of the conclusion chosen by an EEP. With the previous notations, the value of the global action and its quality is given by:

$$A(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i(\mathbf{x}) \times a[i, k[i]] \tag{40}$$

$$\hat{Q}(\mathbf{x}, A(\mathbf{x})) = \sum_{i=1}^{N} \alpha_i(\mathbf{x}) \times q[i, k[i]] \tag{41}$$

7.4.1 Choice of conclusions

The implementation of fuzzy Q-learning implies several competing conclusions by rule. Contrary to what happens for supervised learning, conclusions are not modified: they are analyzed during the exploration phase and their q-values are updated. In the exploitation phase, the greedy policy only takes the best conclusion among the proposed ones: their initial choice is therefore important. Three cases are possible, according to the available *a priori* knowledge.

No knowledge
: It is the most unfavorable case. Let y_{mins} and y_{max} be the lower and upper bounds for a conclusion[8]. The J conclusions are equally distributed in the interval $[y_{min}, y_{max}]$.

Imprecise knowledge
: The knowledge can often be expressed qualitatively by statements such as "turn *slightly* on the right" or "*around* 100". In this case, conclusions must provide different *interpretations* of terms "*slightly*" or "*around*".

Precise knowledge
: Some rules can be exactly known. For example, the central rule of a Proportional-Integral (PI) fuzzy controller is: "if error is zero and its variation is zero then the change in control is zero". Here, it is not necessary to explore: it is sufficient to propose J equal conclusions (here, $a[i,j] = 0$, $j = 1$ to J). This artifice permits to not modify the structure of the algorithm.

This different cases can be bound according to the rules and/or at different stages of learning: in case of no knowledge, the first passage can begin with the full interval $[y_{min}, y_{max}]$. As learning progresses, we

[8] These bounds often correspond to known physical constraints.

progressively pass from no knowledge to imprecise knowledge and so forth.

In any case, it is important to initialize the q-values to zero, in order to not to distort the training. Let's take the example of a mobile robot for which a state S_i is dangerous (e.g. going straight ahead toward an obstacle). Let's suppose that, in this state, an action permits to avoid the danger. If this action is assigned to have an artificially positive q-value, the value of state S_i will be increased paradoxically (see equation 20 or 42), that, by ricochet, will encourage actions driving to this dangerous state...

7.4.2 Algorithm

At time step t, after action $A(\mathbf{x}_t)$, the new state is $\mathbf{x}_{t+1}$. The value of this state (see for example equation 20) corresponds to the quality of the best action that can infer the FIS then. It is immediate to show that this action is the one that is given by the greedy policy. Let $q[i, max[i]]$ be the maximum q-value for the rule i, the value of the state becomes:

$$V_t(\mathbf{x}_{t+1}) = \sum_{i=1}^{N} \alpha_i(\mathbf{x}_{t+1}) \times q[i, max[i]] \tag{42}$$

The temporal difference error can be computed:

$$\triangle Q = \beta\{r_t + \gamma V_t(\mathbf{x}_{t+1}) - \hat{Q}(\mathbf{x}_t, A(\mathbf{x}_t))\} \tag{43}$$

and the elementary quality are immediately updated by:

$$\triangle q[i,j] = \begin{cases} \epsilon \times \triangle Q \times \alpha_i(\mathbf{x}_t) & \text{if } j = k[i] \\ 0 & \text{otherwise} \end{cases} \tag{44}$$

7.4.3 Trace of eligibility

The trace of eligibility, introduced in sections 4 and 5.2, permits to take into account the pairs (states, actions) recently visited: the more such a pair is near of the receipt of a reward or a punishment, the more the corresponding quality will be affected.

It is very easy to adapt the equation 26, that corresponds to the replacing eligibility, in the FIS context. Let $e[i,j]$ be the eligibility of conclusion j used in rule i. This eligibility is updated after the choice of conclusions by the EEP:

$$e[i,j] = \begin{cases} \alpha_i(\mathbf{x}_t) & \text{if } j = k[i] \\ \gamma\lambda e[i,j] & \text{otherwise} \end{cases} \tag{45}$$

The updating of elementary quality (equation 44) becomes:

$$\triangle q[i,j] = \epsilon \times \triangle Q \times e[i,j] \tag{46}$$

The detailed algorithm is presented in table 1. It calls two remarks.

1. The update of q-values for conclusions used in the state $\mathbf{x}_t$ is achieved in the state $\mathbf{x}_{t+1}$. That implies to memorize truth values, $(\alpha_i(\mathbf{x}_t))_{i=1}^N$. It is made automatically by the term $e[i,j]$, even though the eligibility is zero ($\lambda = 0$). However, in the case without eligibility, one can replace the matrix $(e[i,j])_{i,j}$ by a vector v defined by $v_i \leftarrow \alpha_i(\mathbf{x}_t)$ in the stage 5 of the algorithm and reused in the stage 10.

2. The implementation requires to fix the following parameters:

   - the discount factor, γ, $0 \leq \gamma \leq 1$, (equation 43),
   - the learning rate, β, used to update $\hat{Q}(\mathbf{x}_t, A(\mathbf{x}_t))$, (equation 43),
   - the learning rate, ϵ, used to update q-values, (equation 44 or 46),
   - the factor of eligibility, λ, if used (equation 45),
   - the parameter for exploration, for example the probability, P, to choose the action of best quality in the pseudo-stochastic method (section 6.2.1).

1. $t = 0$, $e[i, j] = 0$, observe the state $\mathbf{x}_t$.
2. For every rule, i, compute $\alpha_i(\mathbf{x}_t)$
3. For every rule, i, choose $k[i]$ with an EEP.
4. Compute $A(\mathbf{x}_t)$ (eq. 40)
   and its corresponding quality $\hat{Q}(\mathbf{x}_t, A(\mathbf{x}_t))$ (eq. 41).
5. Compute $e[i, j]$ (eq. 45)
6. Apply the action $A(\mathbf{x}_t)$. Observe the new state, $\mathbf{x}_{t+1}$.
7. Receive the reinforcement, r_t.
8. Compute $\alpha_i(\mathbf{x}_{t+1})$.
9. Compute $\triangle Q$ (eq. 42 and 43).
10. Update the parameters $q[i, j]$ (eq. 46)
    $$q[i, j] \leftarrow q[i, j] + \triangle q[i, j]$$
11. $t \leftarrow t + 1$, go to 3.

Table 1: Algorithm for knowledge extraction

The examination of equations 43 and 44 or 46 shows that parameters β and ϵ are redundant: one can eliminate β (taking $\beta = 1$). It remains four parameters, whose usual values are $0.9 \leq \gamma \leq 0.99$, $0.001 \leq \epsilon \leq 0.1$, $0.9 \leq \lambda \leq 0.95$ and $0.5 \leq P \leq 0.99$. One can find a systematic survey of these parameters in [16].

8 Conclusion

This paper presents the bases of RL and gives some key points for implementation of Q-Learning, one of the most popular RL algorithms.

The last part is devoted to the synergy between RL and Fuzzy Logic, as a way to exploit or interpret the available knowledge. A more detailed review is in preparation.

References

[1] Balch T., "Learning Roles:Behavioral Diversity in Robot Teams", *AAAI Workshop on Multiagent Learning*, 1997.

[2] Barto A., Sutton R., Anderson C., "Neuronlike elements can solve difficult learning control problems", *IEEE Trans. on SMC*, Vol. 13, September 1983.

[3] Barto A., Sutton R., Watkins C., "Learning and sequential decision making", in *Learning and Computational Neuroscience*, MIT Press, Cambridge, 1990.

[4] Bellman R., *Dynamic Programming*, Princeton University Press, 1957.

[5] Beom H., Cho H., "A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning", *IEEE Trans. on SMC*, Vol. 25, March 1995.

[6] Berenji H.R., Khedkar P., "Fuzzy Rules for Guiding Reinforcement Learning", *Proc. of Int. Conf. IPMU'92*, Mallorca, Spain, 1992.

[7] Berenji H.R., "Q-Learning for fuzzy rules", *Proc. of IJCAI'93*, Chambry, 1993.

[8] Bonarini A., "Learning behaviors represented as Fuzzy Logic Controllers", *Proc. of EUFIT'94*, Aachen, September 94.

[9] Caironi P., Dorigo M., Training Q-agents, Tech. Rep. IRIDIA, n 94/14, Universit Libre de Bruxelles, 1994.

[10] Clouse J.A., Utgloff P.E., "A teaching method for reinforcement learning", *Proc. of* 9^{th} *Workshop on Machine Learning ML'92*, 1992.

[11] Garcia F., Ndyaye S., "Apprentissage par renforcement en horizon fini I : comparaison du Q-Learning et du R-Learning", *Apprentissage automatique*, Editions Herms, 1999.

[12] Glorennec P.Y., "Fuzzy Q-Learning and Dynamical Fuzzy Q-Learning", *Proc. of FUZZ-IEEE'94*, Orlando, juin 1994.

[13] Glorennec P.Y., Jouffe L., "Fuzzy Q-Learning", *Proc. of FUZZ-IEEE'97*, Barcelona, July 1997.

[14] Glorennec P.Y., *Algorithmes d'apprentissage pour systmes d'infrence floue*, Editions Herms, 1999.

[15] Homaifar A., McCormick E., ¡¡Simultaneous Design of Membership Functions and Rule Sets for Fuzzy Controllers Using Genetic Algoritms¿¿, *IEEE Trans. on Fuzzy Systems*, Vol. 3, n 2, May 1995.

[16] Jouffe L., "Fuzzy inference system learning by reinforcement methods", *IEEE Trans. on SMC*, Part C, Vol. 28 (3), August 1998.

[17] Lin L-J., "Self-improvement based on reinforcement learning, planning and teaching", *Proc. of* 8^{th} *Workshop on Machine Learning ML'91*, 1991.

[18] McCallum R.A., "Using transitional proximity for faster reinforcement learning", *Proc. of* 9^{th} *Workshop on Machine Learning ML'92*, 1992.

[19] Meuleau N., Le dilemme exploration/exploitation dans les systmes d'apprentissage par renforcement, thse de l'Universit de Caen, 1996.

[20] Munos R., Apprentissage par renforcement, tude du cas continu, thse l'EHESS, 1997.

[21] Ndyaye S., Apprentissage par renforcement en horizon fini : application la gnration de rgles pour la conduite de cultures, thse de l'Universit P. Sabatier, Toulouse, 1999.

[22] Nowé A., "Fuzzy reinforcement learning: an overview", *Advances in fuzzy theory & technology*, P. Wang (Ed.), 1995.

[23] Peng J., Williams R.J., "Incremental multi-step Q-learning", *Machine Learning*, 22, p. 283-290, 1996.

[24] Piché S., "Steepest descent algorithms for NN controllers and filters", *IEEE Trans. on NN*, vol.5 no. 2, march 1994.

[25] Rummery G.A., Niranjan M., On-line Q-Learning using connexionist systems, Tech. Rep. CUED/F-INFENG/TR 156, Cambridge Universit Engineering Department, 1994.

[26] Schwartz A., "A reinforcement learning method for maximizing undiscounted rewards", *Proc. of Int. Conf. on Machine Learning*, 1993.

[27] Sutton R.S., "Learning to predict by the method of temporal differences", *Machine Learning*, 3, p. 9-44, 1989.

[28] Sutton R.S., Barto A.G., *Introduction to reinforcement learning*, MIT Press/Bradford Books, Cambridge, MA, 1998.

[29] Thrun S.B., Mller K., "Active exploration in dynamic environments", *Advances in Neural Information Processing Systems*, Morgan Kaufmann, San Mateo, CA, 1992.

[30] Touzet P., "Neural reinforcement learning for behavior synthesis", *Proc. of CESA'96, IMACS Multi-conference*, Lille, July 1996.

[31] Tsitsiklis P., "Asynchronous Stochastic Approximation and Q-learning", *Machine Learning*, 16, p. 185-202, 1994.

[32] Watkins C., Learning from delayed rewards, PhD Thesis, University of Cambridge, England, 1989.

[33] Watkins C., Dayan P., "Q-learning", *Machine Learning*, 8, p. 279-292, 1992.

[34] Whitehead S.D. “A complexity analysis of cooperative mechanisms in reinforcement learning”, *in Proc. of the 9^{th} AAAI Conf.*, 1991.

[35] Whitley D., Dominic S., Das R., Anderson C., “Genetic reinforcement learning for neurocontrol problems”, *Machine Learning*, 13, p. 259-284, 1993.

[36] Wyatt J., Exploration and inference in learning from reinforcement, PhD, University of Edinburgh, 1997.