Neurocomputing 72 (2009) 1494–1507

Contents lists available at ScienceDirect

Neurocomputing

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

$\mathcal{2L}_2$, a simple reinforcement learning scheme for two-player zero-sum Markov games ☆

Benoît Frénay [a,*], Marco Saerens [b]

[a] Machine Learning Group, EPL/ELEC/DICE, Université catholique de Louvain, Place du Levant, 3 (office a.188.30), 1348 Louvain-la-Neuve, Belgium
[b] Machine Learning Group, ESPO/LSM/ISYS, Université catholique de Louvain, Place des Doyens, 1 (office b.108), 1348 Louvain-la-Neuve, Belgium

ARTICLE INFO

Available online 12 January 2009

Keywords:
Reinforcement learning
Q-Learning
Markov games
Two-player zero-sum games
Multi-agent

ABSTRACT

Markov games is a framework which can be used to formalise n-agent reinforcement learning (RL). Littman (Markov games as a framework for multi-agent reinforcement learning, in: Proceedings of the 11th International Conference on Machine Learning (ICML-94), 1994.) uses this framework to model two-agent zero-sum problems and, within this context, proposes the minimax-Q algorithm. This paper reviews RL algorithms for two-player zero-sum Markov games and introduces a new, simple, fast, algorithm, called $\mathcal{2L}_2$. $\mathcal{2L}_2$ is compared to several standard algorithms (Q-learning, Minimax and minimax-Q) implemented with the `2ash` library written in `Python`. The experiments show that $\mathcal{2L}_2$ converges empirically to optimal mixed policies, as minimax-Q, but uses a surprisingly simple and cheap updating rule.

1. Introduction

In opposition to supervised learning, reinforcement learning (RL) allows modelling problems for which it is impossible to provide a learning set to the agent: it requires no teacher, no learning pairs and, in fact, no prior knowledge. The agent learns from direct interaction with its environment and when it succeeds, partially or not, it receives a numerical reward. Thus, the agent proceeds by trial-and-error, alternating exploration of the problem and exploitation of its experience, and slowly learns the actions that lead to the rewards.

RL has been applied with success to many problems: backgammon [36], elevator dispatching [12], dynamic channel allocation [29], investment decision making [25], packaging in a food processing industry [19], space shuttle payload processing for NASA [45], etc. We think that its ability to tackle problems where even human experts lack the knowledge necessary for traditional approaches can be an advantage.

This paper addresses specifically two-player zero-sum games. Littman has developed an efficient and optimal algorithm [22,34], called minimax-Q, which has to solve a linear optimisation problem at each step. We propose a new algorithm $\mathcal{2L}_2$, and a variant $\mathcal{2L}_2^+$, which also achieves optimality, but is much simpler and faster in terms of computational time. Each algorithm mentioned in this paper has been implemented with the `2ash` library, a `Python` generic framework for RL in n-player Markov games available for free download at [1].

1.1. Related work and contributions

Sutton and Barto's book [33] is entirely devoted to one-agent RL and introduces the Q-learning algorithm [42,43], which is also discussed in e.g. [17,24,28]. It uses Markov decision processes (MDPs), which are studied in details in [13,30,37]. Moreover, a dynamic programming approach with policy iteration and value iteration is addressed in [4,5].

Two-agent or n-agent RL are described in Littman's paper [22] which has several contributions:

- it compares three different models for two-agent problems (MDPs, matrix games [32] and Markov games);
- it proposes the minimax-Q algorithm, which exploits the links between RL and game theory;
- it demonstrates the respective strengths of Q-learning and minimax-Q on a soccer game.

The work in [34] adopts a more theoretical approach and, in particular, studies the convergence of several RL algorithms, including Q-learning and minimax-Q, whereas [39] proposes several extensions to the minimax-Q algorithm. Interesting references to get a grip on game theory are [6,27,32,41] and

☆ A short, preliminary, version of this paper was published in the proceedings of the ESANN conference [15].

* Corresponding author.
E-mail addresses: benoit.frenay@uclouvain.be (B. Frénay), marco.saerens@uclouvain.be (M. Saerens).

0925-2312/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2008.12.022

mathematical tools for solving Markov games are covered by Filar and Vrieze [14].

This work addresses two-player zero-sum games with RL and has three contributions:

- it reviews two-agent RL in the unified framework of Markov games (Section 2) and the existing Q-learning, Minimax [28] and minimax-Q algorithms (Section 3);
- it proposes a new simple algorithm, $\mathcal{2L}_2$, and its variant $\mathcal{2L}_2^+$ (Section 3);
- it compares theoretically and experimentally these algorithms (Sections 3–5).

The interest of $\mathcal{2L}_2$ (and especially $\mathcal{2L}_2^+$) lies in its cheap and easy-to-implement policy update rule: whereas Littman's algorithm solves a linear optimisation problem, these algorithms involve only arithmetical operations and exponential evaluations. As a matter of fact, $\mathcal{2L}_2$ and $\mathcal{2L}_2^+$ appear to be faster in terms of computational time (see Section 4).

Another algorithm which is not addressed in this paper is the *R*-MAX algorithm [8]. It starts with an initial optimistic model of the environment where any action performed in any state yields a maximal reward R_{max}. Then, the model is updated according to the following rule: if some actions have been observed K_1 times in a given state, this state becomes known and the model is updated with the empirical rewards and transition frequencies. It can be shown [8] that K_1 can be chosen *a priori* to obtain guarantees on the algorithm's convergence in terms of computational time.

In this paper, we will not consider the possibility to use approximators for RL. For example, the replacement of the Q function by a function approximator would certainly accelerate the learning process, but we would not be able to provide strong *a priori* guarantees on the algorithms' performances (see e.g. [4,7,21,39] for more details).

2. Basic framework and notations

This section reviews two-agent RL in the unified framework of Markov games. Let us consider [22,33,42,43] the learning process of an agent $\mathcal{A}g$ acting on an environment $\mathcal{E}$, possibly simultaneously with others agents which may be *friends* or *foes*.

The *environment* $\mathcal{E}$ is characterised at time t by its (observable) *state* X_t which is a measurement on some features of $\mathcal{E}$. For simplicity, $\mathcal{E}$ is restricted to be discrete-time, discrete-state, static and observable with no uncertainty [28]. The probabilistic behaviour of $\mathcal{E}$ can be predicted from its current state, if the *Markov property* and the *stationary assumption* both hold. The former requires that the probability of the next state of $\mathcal{E}$ is only conditioned by its current state, whereas the latter requires that $\mathcal{E}$'s behaviour is time-homogeneous. If the state contains enough information, many environments can be approximated by *Markovian models* such as Markov chains, MDPs or Markov games.

The *agent* $\mathcal{A}g$ interacts with its environment $\mathcal{E}$: it perceives X_t *via sensors* and modifies $\mathcal{E}$ *via* actions done by *actuators*. *The* agent-environment boundary represents the limit of the agent's absolute control [33]; from an external point of a view, an agent is just a function mapping a sequence of states to an action. The goal of an agent is to realise a certain *task* $\mathcal{T}k$, e.g. to reach a power supply, to clean a room encumbered by some garbage or to win a chess game.

As mentioned in [28], tasks can be *episodic* if the agent–environment interaction breaks naturally into subsequences called *episodes*, or *continuing* if the agent-environment interaction goes on forever. Given an environment $\mathcal{E}$, a task $\mathcal{T}k$ and some prior knowledge, the agent has to find the right action for each possible state sequence $(X_0, \ldots, X_t)$.

In fact, in many situations, the right action is just unknown, but one can use *RL* (see e.g. [11,17,24,33]) which assumes no teacher, no learning pairs and in fact *no prior knowledge* about the environment. The learning agent learns from *direct interaction* with its environment: it is only given a task to achieve and when it succeeds, even partially, it receives a *reward*, i.e. a numeric value to reward him. The agent proceeds by *trial–error* and slowly learns the actions that lead to the rewards.

2.1. Reward signal and return

In RL, a learning problem is specified by an environment $\mathcal{E}$, a reward signal r associated to a task $\mathcal{T}k$ and a return R. The *reward signal* r is a bounded numeric value indicating the *intrinsic desirability* of a state of $\mathcal{E}$ for the success of $\mathcal{T}k$. It can be compared to *fear* and *pleasure* in animal psychology; the agent gets high positive rewards when the environment enters good states and low or negative rewards when it enters bad states.

A learning agent will try to maximise the *accumulated amount of rewards* quantified by the *return* R, a function mapping the sequence of rewards received when starting from $X_t = x_t$ to a (possibly negative) numeric value R_t. In this paper, we use the *infinite-horizon discounted return* [4,30,33], i.e., respectively, for episodic tasks (limited in time) and for continuing tasks (unlimited)

$$R_t = \sum_{i=0}^{T} \gamma^i r_{t+i} \quad \text{and} \quad R_t = \sum_{i=0}^{+\infty} \gamma^i r_{t+i}, \tag{1}$$

where T is the time remaining until the end of the episodic task and $\gamma \in]0,1[$ is the *discount rate* implying that the sooner is the reward, the better is the return. Intuitively, γ can also be interpreted as the probability to continue the task [2,22].

2.2. Markov games

In RL, *n-agent environments* can be modelled by *n-player Markov games* with n distinct reward signals and which transitions depends on the n *simultaneous* actions of the agents. In this paper, $n = 2$ and the interest of both players (the *agent* $\mathcal{A}g$ and its *opponent* $\mathcal{O}pp$) are *opposite*, i.e. we treat *zero-sum* games.

A *two-player zero-sum Markov game* $\mathcal{MG}$ [14,22] is a tuple $\langle \mathcal{S}, s_{init}, \mathcal{D}, \mathcal{A}, \mathcal{O}, A, O, T, r \rangle$ where $\mathcal{S}$ is the set of states, $s_{init} \in \mathcal{S}$ is the unique initial state, $\emptyset \subseteq \mathcal{D} \subseteq \mathcal{S} \backslash \{s_{init}\}$ is the set of absorbing states, $\mathcal{A}$ is the set of the agent's actions, $\mathcal{O}$ is the set of the opponent's actions, $A : \mathcal{S} \to \mathcal{A}$ gives for each state the set of available actions for the agent, $O : \mathcal{S} \to \mathcal{O}$ gives for each state the set of available actions for the opponent, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \times \mathcal{S} \to \Re^+$ is the transition probability distribution for the whole environment such that

$$T_{s_i,a,o,s_j} = \Pr(X_{t+1} = s_j | X_t = s_i, a, o) \tag{2}$$

and $r : \mathcal{S} \times \mathcal{A} \times \mathcal{O} \to \Re$ gives the reward $r_s(a,o)$ obtained by the agent in s if it chooses a and its opponent chooses o.

In this paper, $\mathcal{S}$, $\mathcal{D}$, $\mathcal{A}$ and $\mathcal{O}$ are finite sets and r is deterministic. s_{init} is introduced to simplify further developments and cannot be reached from other states:

$$\forall s \in \mathcal{S}, a \in A(s), o \in O(s) : T_{s,a,o,s_{init}} = 0. \tag{3}$$

If the considered environment has more than one initial state, s_{init} is added as an additional (virtual) state put somewhere outside the real environment $\mathcal{S} \backslash s_{init}$ and leading to the true initial states, which are called the *entry states*. As for them, the *absorbing states* allow to unify episodic and continuing tasks [33]; indeed,

the return can always be expressed as

$$R_t = \sum_{i=0}^{+\infty} \gamma^i r_{t+i} \tag{4}$$

if we consider that an episode is ended when an absorbing state is reached and that

$$\forall d \in \mathscr{D}, a \in A(d), o \in O(d) : r_d(a,o) = 0. \tag{5}$$

As mentioned in [22], we can consider Markov games as a generalisation of both MDPs ($|\mathscr{O}| = 1$) and matrix games ($\mathscr{S} \backslash \mathscr{D} = \{s_{init}\}$). In essence, Markov games are *simultaneous games*, but they can easily model *alternate games* [32].

2.3. Policies

The behaviours of the agent $\mathscr{A}g$ and its opponent $\mathscr{O}pp$ are, respectively, described by the policies π and σ. A *policy* is a formalisation for the prescription of the agents for taking actions [22,37]. In this paper, we consider *stationary policies* [37] which depend only on the current state. For an agent $\mathscr{A}g$ acting on an environment $\mathscr{E}$, a *pure policy* $\pi : \mathscr{S} \to \mathscr{A}$ associates to each state s the action $\pi(s)$ and a *mixed policy* $\pi : \mathscr{S} \times \mathscr{A} \to \Re^+$ associates to each state s and action $a \in A(s)$ the probability $\pi_s(a)$ to choose a.

Note that we consider only *rational* opponents [6,27,32] which do not *cooperate* in zero-sum games. Indeed, the reward gained by the agent after any action is always the opposite of its opponent's. This assumption is called the *rational player hypothesis*.

2.4. Optimality and the value function V

Now that both the environment's and the agent's behaviour are formalised, the following of this section will introduce the RL problem.

In RL, we are interested in agents maximising the *expected return* for each state. This expected return is called the value function V and aims to *estimate* how good *it is for the agent to be in a given state in terms of future rewards that can be expected* [33]. In order to compute V, we need to determine the opponent's optimal policy and we have to consider the worst possible opponent: a rational player who knows the agent's entire policy π.

Game theory [6,27,32] tells us that, in zero-sum games, any rational opponent tries to minimise the agent's expected return, so that we can define V as the worst expected return for the agent, given its current policy. Therefore, the *value function* $V_\pi : \mathscr{S} \to \Re$ for an environment $\mathscr{E}$ and an agent $\mathscr{A}g$ with a policy π aiming to fulfil a task $\mathscr{T}k$ against a rational opponent $\mathscr{O}pp$ gives, for each state s, the expected return if $\mathscr{A}g$ acts on $\mathscr{E}$ according to π starting from s while $\mathscr{O}pp$ acts rationally, i.e.

$$V_\pi(s) = \min_\sigma \underset{\pi,\sigma}{E}\left\{ \sum_{i=0}^{+\infty} \gamma^i r_{t+i} \middle| X_t = s \right\}. \tag{6}$$

The corresponding (non-necessary unique) *optimal policy* π^* for $\mathscr{A}g$ maximises $V_\pi(s)$ for each state s of $\mathscr{E}$, i.e.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\ V_\pi(s) \quad \forall s \in S \tag{7}$$

with V_{π^*} called the *optimal value function* V^*. $V_\pi(s)$ and $V^*(s)$ are, respectively, called the *value of the state s* (with respect to π) and the *optimal value of the state s*.

Now we can define the RL problem. For an environment $\mathscr{E}$ with the Markov property and the stationary assumption, a *two-agent RL problem* is defined by the tuple $\langle \mathscr{E}, \mathscr{T}k, R \rangle$. It consists of finding the optimal policy π^* associated with $\mathscr{T}k$ and the return R. A solution to a two-agent RL problem provides for each possible state s of $\mathscr{E}$ its optimal value $V^*(s)$ and a prescription for taking actions in s optimally, provided that the opponent is rational. Note that optimal policies are not necessarily unique [6].

3. Algorithms for two-agent RL problems

This section discusses several standard RL algorithms and proposes a new algorithm, $\mathscr{2}\mathscr{L}_2$, in order to solve two-agent games from the perspective of the agent $\mathscr{A}g$. In [26], it is shown that the value iteration algorithm [33] (which requires a model of the environment) adapted to Markov games does converge. However, many real-world problems lack a complete description of the associated Markov game. In that case, T and r are unavailable and the learning is rather performed online, i.e. in direct interaction with the environment, with temporal difference algorithms [24,33].

The algorithms discussed in this paper are modelled on the general template Algorithm 1 which solves a RL problem $\langle \mathscr{E}, \mathscr{T}k, R \rangle$ with *no prior knowledge* of $\mathscr{E}$ and $\mathscr{T}k$. The learning consists of *epochs* $\langle s, a, o, r, s_s \rangle$: the agent perceives a state s, takes an action a, observes the action o performed by its opponent, receives in return a reward r, perceives the resulting state s_s and then learns from it. With temporal difference, prior knowledge is no longer necessary for the learning process. Moreover, the agent learns with each epoch, so that it is suitable for both episodic and continuing tasks: the agent does not have to wait for a full episode to learn.

Algorithm 1. Online Markov games algorithms template.

Input: A RL problem $\langle \mathscr{E}, \mathscr{T}k, R \rangle$
Output: $\widehat{V^*}$,$\widehat{\pi^*}$
1. Initialise V or/and miscellaneous data structures
2. $s \leftarrow$ current state of $\mathscr{E}$
3. **repeat**
4. Choose and take action a
5. $o, r, s_s \leftarrow$ action taken by $\mathscr{O}pp$, gained reward, resulting state of $\mathscr{E}$
6. Learn from the epoch $\langle s, a, o, r, s_s \rangle$
7. $s \leftarrow s_s$
8. **until** some convergence criterion is satisfied
9. **return** V_π,π

Note that temporal difference methods introduce a new problem: the *exploration–exploitation dilemma* [33] which is not specific to these methods, but is inherent to RL. The agent has two important but contradictory tasks to achieve in order to find an optimal policy: the *exploration* of its environment and the *exploitation* of its acquired knowledge. These two tasks are intrinsically opposite; at the same time, the agent must investigate unknown (possibly unsuccesfull) paths and focus in the right direction, i.e. head toward paying states. In Algorithm 1, the exploration–exploitation balance is ensured by the instruction *choose and take action a* and a popular (simple) balance scheme called ε-*greedy* [33] consists to choose randomly between exploration or exploitation for the current epoch. The exploration probability is called the *exploration degree* ε.

In this paper, we discuss five algorithms modelled on the template Algorithm 1:

- the standard one-agent simple *Q-learning* [42,43];
- *Minimax* [28], originated from game theory, which insures to obtain at least the maximin value;
- *minimax-Q* [22] which uses linear programming;
- $\mathscr{2}\mathscr{L}_2$ and $\mathscr{2}\mathscr{L}_2^+$, our new algorithms which use a gradient-ascent scheme to optimise $V_\pi(s_{init})$.

In these algorithms discussed below, the only differences lie in the implementation of *learn from the epoch* $\langle s, a, o, r, s_s \rangle$ which is therefore the only part made explicit.

3.1. The evaluation function Q and the Bellman equations for Markov games

In order to implement Algorithm 1 by specific algorithms, we need to introduce an additional quantity. The *evaluation function* $Q_\pi : \mathscr{S} \times \mathscr{A} \times \mathscr{O} \to \Re$ for an environment $\mathscr{E}$ and an agent $\mathscr{A}g$ with policy π aiming to fulfil a task $\mathscr{T}k$ against a rational opponent $\mathscr{O}pp$ gives, for each state s and actions $a, o \in A(s) \times O(s)$, the expected return if $\mathscr{A}g$ and $\mathscr{O}pp$, respectively, do a and o in s and then act according to π and a rational policy, i.e.

$$Q_\pi(s,a,o) = \min_\sigma \mathop{E}_{\pi,\sigma}\left\{ \sum_{i=0}^{+\infty} \gamma^i r_{t+i} \middle| X_t = s, a, o \right\}. \tag{8}$$

This quantity is fundamental because we can approximate the Q *values* with Q-learning and similar algorithms, even if we do not know T and r. Afterward, the V *values* can be deduced: V and Q functions are related and can be expressed recursively. If we introduce

$$\overline{Q}_\pi(s,o) = \sum_{a \in A(s)} \pi_s(a) Q_\pi(s,a,o), \tag{9}$$

which is the return expected in s with the current policy π if the opponent chooses o, the *Bellman equations for Markov games* [21,22,33] are for each $s \in \mathscr{S} \backslash \mathscr{D}$

$$V_\pi(s) = \min_{o \in O(s)} \overline{Q}_\pi(s,o) \tag{10}$$

and for each $s \in \mathscr{S} \backslash \mathscr{D}, a \in A(s), o \in O(s)$

$$Q_\pi(s,a,o) = r_s(a,o) + \gamma \sum_{s_s \in S} T_{s,a,o,s_s} \min_{o_s \in O(s_s)} \overline{Q}_\pi(s_s, o_s). \tag{11}$$

Moreover, the V and Q values are zero for the absorbing states.

Note that in Eq. (11), $r_s(a,o)$ is outside the sum: we only need the reward received upon executing the actions a, o, not the full definition of the reward signal in s. The Bellman equations (10) and (11) will be used by each subsequent algorithm to compute the Q values; they only differ by the hypotheses and the methods they use to compute π^*.

3.2. Simple Q-learning

The next five subsections will describe five algorithms allowing to solve two-player zero-sum Markov games. Although it was originally developed for one-agent environments, the simple Q-learning [42,43,33] algorithm can be used on two-agent environments, considering that the opponent is part of the environment.

3.2.1. Reducing two-agent environments

If we consider the Markov game $\mathscr{MG} = \langle \mathscr{S}, s_{init}, \mathscr{D}, \mathscr{A}, \mathscr{O}, A, O, T, r \rangle$ and $\mathscr{O}pp$ which actual policy σ is not necessarily rational, we can model both $\mathscr{MG}$ and $\mathscr{O}pp$ as the components of an one-agent environment by integrating the opponent in the environment. If σ is Markovian and stationary, we obtain a new Markov decision process $\mathscr{MDP} = \langle \mathscr{S}, s_{init}, \mathscr{D}, \mathscr{A}, A, T', r' \rangle$. The transition probability distribution T becomes T' s.t.

$$T'_{s_i,a,s_j} = \sum_{o \in \mathscr{O}(s_i)} \sigma_{s_i}(o) T_{s_i,a,o,s_j} \tag{12}$$

and the reward signal r becomes r' s.t.

$$r'_s(a) = \sum_{o \in \mathscr{O}(s)} \sigma_s(o) r_s(a,o). \tag{13}$$

For one-agent environments, the Bellman equations simplify to

$$V_\pi(s) = \max_{a \in A(s)} Q_\pi(s,a) \tag{14}$$

and

$$Q_\pi(s,a) = r_s(a) + \gamma \sum_{s_s \in S} T_{s,a,s_s} \max_{a_s \in A(s_s)} Q_\pi(s_s, a_s). \tag{15}$$

Note that the agent uses a pure policy and that the opponent's action are not mentioned anymore.

3.2.2. Algorithm

In Q-learning, the Q values are computed *iteratively*. We start with arbitrary initial Q values and, when $\mathscr{A}g$ observes an epoch $\langle s, a, r, s_s \rangle$ ($\mathscr{A}g$ does not care about the opponent's action o), it uses the *update rule*

$$Q_\pi(s,a) \leftarrow (1-\alpha) Q_\pi(s,a) + \alpha \left[r + \gamma \max_{a_s \in A(s_s)} Q_\pi(s_s, a_s) \right], \tag{16}$$

where $\alpha \in]0,1]$ ($\alpha = 1$ for deterministic environments, otherwise α decreases during the learning). It works because *the probability with which this happens is precisely* T_{s,a,s_s} [22]; it corresponds to a stochastic approximation [31] of the actual Q.

Algorithm 2 shows the implementation of *learn from the epoch* $\langle s, a, o, r, s_s \rangle$ in the general template Algorithm 1 for the Q-learning algorithm. If $0 < \gamma < 1$, a raw table initialised with zero values is used to store the Q values, each state-action pair is assumed to be visited infinitely often and the α parameter decreases properly (necessary for stochastic environments only), Q-learning converges to the true V^* and π^* corresponding to the MDP [24,43]. Note that Q-learning will generally find a policy which is not optimal for the original Markov game.

Algorithm 2. Learning part of the Q-learning algorithm.

1. $Q_\pi(s,a) \leftarrow (1-\alpha) Q_\pi(s,a) + \alpha [r + \gamma \max_{a_s \in A(s_s)} Q_\pi(s_s, a_s)]$
2. $\pi(s) \leftarrow argmax_{a \in A(s)} Q_\pi(s,a)$

Q-learning does not consider the rational player hypothesis [32]. Since $\mathscr{O}pp$ is part of the environment, $\mathscr{A}g$ learns to beat him *specifically* and adapts his play to $\mathscr{O}pp$'s *actual playing level*, i.e. the degree of optimality of its policy. But this also means that $\mathscr{A}g$ is *opponent-dependent*. If $\mathscr{O}pp$ changes significantly or is replaced, $\mathscr{A}g$ will not change his course of action, which is most likely not suited anymore; $\mathscr{A}g$ will have to start learning again.

In fact, the playing level of $\mathscr{A}g$ is entirely conditioned by the playing level of $\mathscr{O}pp$. Moreover, if the behaviour of $\mathscr{O}pp$ is not Markovian or not stationary, the resulting one-agent environment is not a MDP and the convergence is no longer ensured. Note that, even if the original environment is deterministic, the embedding of $\mathscr{O}pp$ may produce a stochastic one-agent environment and slow down the learning.

The update rule Eq. (16) must be adapted for the next algorithms. Indeed, as opposed to the simple Q-learning, they consider the environment and the opponent as two distinct entities, i.e. they truly consider two-player zero-sum Markov games. For pure and mixed policies, Eq. (16) becomes, respectively, [22]

$$Q_\pi(s,a,o) \leftarrow (1-\alpha) Q_\pi(s,a,o) + \alpha \left[r + \gamma \max_{a_s \in A(s_s)} \min_{o_s \in O(s_s)} Q_\pi(s_s, a_s, o_s) \right] \tag{17}$$

and

$$Q_\pi(s,a,o) \leftarrow (1-\alpha) Q_\pi(s,a,o) + \alpha \left[r + \gamma \min_{o_s \in O(s_s)} \overline{Q}_\pi(s_s, o_s) \right], \tag{18}$$

where $\alpha \in]0,1]$. Like Eq. (16), the update rules Eqs. (17) and (18) do not explicitly use T, but T regulates again the frequency of the updates during the stochastic approximation process.

3.3. minimax-Q: a linear programming approach

A first approach to solve Markov games, which has been used by Littman [22], is the *linear programming* approach. Here, we introduce and analyse his algorithm, minimax-Q.

3.3.1. The RL problem as a set of matrix games

The key idea of minimax-Q is to formulate the RL problem as a set of *matrix games* [32]. Indeed, for each state $s \in \mathscr{S}$, we can create a matrix game $\mathscr{G}_s$ involving $\mathscr{A}g$ and $\mathscr{O}pp$ in three steps. Firstly, we create a strategy $A_{s,a}$, for each action $a \in A(s)$, which prescribes $\mathscr{A}g$ to take a in s and to follow π thereafter. Secondly, we create a strategy $O_{s,o}$, for each action $o \in O(s)$, which prescribes $\mathscr{O}pp$ to take o in s and to follow a rational policy thereafter. Thirdly, we associate to each couple of strategies $\langle A_{s,a}, O_{s,o}\rangle$ the payoff

$$\Pi(A_{s,a}, O_{s,o}) = Q_\pi(s,a,o). \tag{19}$$

For $\mathscr{G}_s$, the optimal (mixed) strategy for $\mathscr{A}g$ will have the form

$$\sum_{a\in A(s)} \pi_s^*(a) A_{s,a}. \tag{20}$$

In [34], it is shown that, if we solve these $|\mathscr{S}|$ matrix games and use the update rule Eq. (18), the process will converge to V^* and π^*. Note that minimax-Q has to solve a linear pogramming problem at each epoch.

3.3.2. Algorithm

It is well-known that zero-sum matrix games can be viewed as linear programming problems [16,32]: the *optimisation problem* solving $\mathscr{G}_s$ is

$$\left|\begin{array}{ll} \max\limits_{\pi_s} & \min\limits_{o\in O(s)} \sum\limits_{a\in A(s)} \pi_s(a)\Pi(A_{s,a}, O_{s,o}) \\ \text{s.t.} & 0 \leqslant \pi_s(a) \leqslant 1 \quad \forall a \in A(s), \\ & \sum\limits_{a\in A(s)} \pi_s(a) = 1, \end{array}\right. \tag{21}$$

where the objective function is the expected payoff for $\mathscr{G}_s$, i.e. the expected return in s with the policy π. This program is not linear, but if we add a new variable $v_{\mathscr{G}_s}$ and several constraint which prevents $v_{\mathscr{G}_s}$ to exceed $\sum_{a\in A(s)} \pi_s(a)\Pi(A_{s,a}, O_{s,o}) = \overline{Q}_\pi(s,o)$, it becomes

$$\left|\begin{array}{ll} \max\limits_{\pi_s, v_{\mathscr{G}_s}} & v_{\mathscr{G}_s} \\ \text{s.t.} & \overline{Q}_\pi(s,o) \geqslant v_{\mathscr{G}_s} \quad \forall o \in O(s), \\ & 0 \leqslant \pi_s(a) \leqslant 1 \quad \forall a \in A(s), \\ & \sum\limits_{a\in A(s)} \pi_s(a) = 1. \end{array}\right. \tag{22}$$

Eventually, we obtain Littman's minimax-Q algorithm shown in Algorithm 3. It solves one small linear optimisation problem by learning epoch: $\mathscr{G}_s$ is a $|A(s)| \times |O(s)|$ matrix game and is solved by a linear optimisation problem with $|A(s)| + |O(s)|$ variables and $|O(s)| + 1$ constraints in normal form. It can be solved with the simplex algorithm [3] which has a theoretical exponential worst-case time complexity [20], but nevertheless often performs very well on practical problems (for which the number of iterations is usually a small multiple of the problem dimension, see e.g. [38,44]).

Algorithm 3. Learning part of the minimax-Q algorithm.

1. $Q_\pi(s,a,o) \leftarrow (1-\alpha)Q_\pi(s,a,o) + \alpha[r + \gamma \min_{o_s \in O(s_s)} \overline{Q}_\pi(s_s, o_s)]$
2. Solve the linear program associated to $\mathscr{G}_s$

An advantage of Littman's approach is that, at any time, the policy in a given state is locally optimal with respect to the current Q values of the state (which is not necessary the case for gradient-ascent methods like $\mathscr{2L}_2$). But it uses a linear solver, which can be quite heavy, even if the linear programs remain small. Note that the linear programming step has to be repeated at each state-action visit.

3.4. Minimax

For some Markov games, the mixed policies are not always essential to achieve optimality. If we use only pure policies, i.e. $\mathscr{A}g$ uses a *maximin strategy* [32], we will at least obtain $\text{maximin}_{\mathscr{G}_{s_{init}}}$ and, if the rational opponent uses a pure policy as well, we will at most obtain $\text{minimax}_{\mathscr{G}_{s_{init}}}$. Therefore, if

$$\text{maximin}_{\mathscr{G}_{s_{init}}} = \text{minimax}_{\mathscr{G}_{s_{init}}} \tag{23}$$

then mixed policies are not necessary to achieve optimality.

Originated from game theory, the standard Minimax algorithm [32], used in association with the update rule Eq. (18), can be applied to the RL problem when we restrict ourselves to pure policies: we then obtain Algorithm 4. Minimax is a valuable algorithm for alternate games, since pure policies are sufficient for this type of game: it is computationally less demanding than algorithms like minimax-Q or $\mathscr{2L}_2$, which both optimise a mixed policy.

Algorithm 4. Learning part of the Minimax algorithm.

1. $Q_\pi(s,a,o) \leftarrow (1-\alpha)Q_\pi(s,a,o) + \alpha[r + \gamma \max_{a_s in A(s_s)} \min_{o_s \in O(s_s)} Q_\pi(s_s, a_s, o_s)]$
2. $\pi(s) \leftarrow argmax_{a\in A(s)} \min_{o\in O(s)} Q_\pi(s,a,o)$

However, Minimax will give sub-optimal results on a certain class of simultaneous games. For example, let us consider the matrix game $\mathscr{G}$ defined by Table 1. Minimax will learn that it can obtain for sure at least $\text{maximin}_{\mathscr{G}} = 0$ and at most $\text{minimax}_{\mathscr{G}} = +4$, whatever the strategy it chooses. Thus, it will indistinctly produce a policy prescribing A_1, A_2 or A_3. However, the optimal mixed strategy for $\mathscr{G}$ is $\frac{5}{9}A_1 + \frac{4}{9}A_3$ and the value of the game is $v_{\mathscr{G}} = \frac{20}{9}$.

3.5. $\mathscr{2L}_2$: A constrained optimisation approach

Both minimax-Q and Minimax have at least one important drawback: the former involves the resolution of one linear optimisation problem at each epoch, whereas the latter produces sub-optimal policies on games where mixed policies are required

Table 1
Normal form of $\mathscr{G}$ and computation of $\text{maximin}_{\mathscr{G}}$ and $\text{minimax}_{\mathscr{G}}$.

		$\mathscr{O}pp$				
		O_1	O_1	O_3	min	
$\mathscr{A}g$	A_1	0	+3	+4	0	← maximin
	A_2	+2	+4	0	0	← maximin
	A_3	+5	+3	0	0	← maximin
	max	+5	+4	+4		
			↑	↑		
			minimax			

to achieve optimality. We now introduce the main contribution of this paper, $\mathcal{QL}_2$, an alternative algorithm using a gradient-ascent scheme to optimise the agent's mixed policy under the Bellman equations.

3.5.1. Problem and Lagrangian formulation

The key idea of our algorithm is to formulate the RL problem as a constrained optimisation problem (the constraints being the Bellman equations):

$$\left|\begin{array}{ll}\max\limits_{\pi} & V_\pi(s_{init}) \\ \text{s.t.} & V_\pi(s_{init}) = \min\limits_{o\in O(s_{init})} \overline{Q}_\pi(s_{init},o), \\ & Q_\pi(s,a,o) = \begin{cases} r_s(a,o) + \gamma \sum\limits_{s_s\in\mathcal{S}} \left[T_{s,a,o,s_s} \min\limits_{o_s\in O(s_s)} \overline{Q}_\pi(s_s,o_s)\right] \\ \quad \forall s\in\mathcal{S}\backslash\mathcal{D}, a\in A(s), o\in O(s), \\ 0 \quad \forall s\in\mathcal{D}, a\in A(s), o\in O(s). \end{cases}\end{array}\right. \tag{24}$$

Note that we optimise $V_\pi(s_{init})$ instead of each state value $V_\pi(s), s\in\mathcal{S}$: intuitively, the goal of an agent in a RL problem is to *estimate the value of the problem*, i.e. the expected return for an optimal policy

$$V^*(s_{init}) = \min_{\sigma} \underset{\pi^*,\sigma}{E}\{R_0|X_0 = s_{init}\}. \tag{25}$$

Therefore, it is sufficient to optimise $V_\pi(s_{init})$ in order to optimise the expected return of the agent which always starts in s_{init}. Replacing $V_\pi(s_{init})$, we obtain the Lagrangian

$$\begin{aligned}\mathcal{L}_\pi = & \min_{o\in O(s_{init})} \overline{Q}_\pi(s_{init},o) \\ & + \sum_{s\in\mathcal{S}\backslash\mathcal{D}} \sum_{a\in A(s)} \sum_{o\in O(s)} \lambda_{s,a,o}[Q_\pi(s,a,o) - \\ & \left[r_s(a,o) + \gamma \sum_{s_s\in\mathcal{S}} \left[T_{s,a,o,s_s} \min_{o_s\in O(s_s)} \overline{Q}_\pi(s_s,o_s)\right]\right]\Big] \\ & + \sum_{s\in\mathcal{D}} \sum_{a\in A(s)} \sum_{o\in O(s)} \lambda_{s,a,o}[Q_\pi(s,a,o) - 0],\end{aligned} \tag{26}$$

where λ are Lagrange multipliers. Note also that the variables $\pi_s(a)$ are constrained in order to satisfy

$$\begin{aligned} & 0\leqslant\pi_s(a)\leqslant 1 \quad \forall a\in A(s), \\ & \sum_{a\in A(s)} \pi_s(a) = 1. \end{aligned} \tag{27}$$

3.5.2. The softmin$^\phi$ operator

In Eq. (26), $\mathcal{L}_\pi$ is expressed using the min operator which has *no analytic derivatives* and is *not differentiable* at break-points. We therefore introduce [9,10,18,33]

$$\underset{i\in 1..n}{\text{softmin}}\, f_i = \frac{\sum_{i=1}^n e^{\phi f_i} f_i}{\sum_{i=1}^n e^{\phi f_i}}, \tag{28}$$

where $\phi\in]-\infty,0]$, which has the remarkable property

$$\lim_{\phi\to-\infty} \underset{i\in 1..n}{\text{softmin}}\, f_i = \min_{i\in 1..n} f_i. \tag{29}$$

Not only we can approximate the min operator by softmin$^\phi$ with finite values of ϕ, but we also obtain the approximation

$$\frac{\partial \min_{i\in 1..n} f_i}{\partial f_k} \approx \frac{\partial\, \text{softmin}_{i\in 1..n} f_i}{\partial f_k} = \frac{e^{\phi f_k}\left[1+\phi(f_k - \text{softmin}_{i\in 1..n} f_i)\right]}{\sum_{i=1}^n e^{\phi f_i}}, \tag{30}$$

which can be evaluated, even at break points. This is very important, since we do not know their locations *a priori*. Note that when $\phi = 0$, $\text{softmin}^\phi_{i\in 1...n} f_i = mean_{i\in 1...n} f_i$. Therefore, ϕ can be tuned to use softmin$^\phi$ as an intermediate between the *mean* and min operators: values of ϕ close to zero bring softmin$^\phi$ closer to *mean* whereas more negative values bring softmin$^\phi$ closer to min. In practice ϕ values must be chosen in order to keep the value of the ϕf_i expressions small enough ($\phi\geqslant -100$).

3.5.3. Local θ update rule

We further define the transformation [9,10,18,33]

$$\pi_s^\theta(a) = \frac{e^{\theta_s(a)}}{\sum_{b\in A(s)} e^{\theta_s(b)}}, \tag{31}$$

where $\theta_s(a)\in]-\infty,+\infty[$ is a parameter such that the bigger $\theta_s(a)$, the bigger $\pi_s^\theta(a)$ (and *vice versa*). The advantages of π^θ are two-fold:

- its derivatives are non-constant, but still simple enough:

$$\frac{\partial \pi_s^\theta(a)}{\partial \theta_s(a)} = \pi_s^\theta(a)(1-\pi_s^\theta(a)), \tag{32}$$

$$\frac{\partial \pi_s^\theta(a)}{\partial \theta_s(b)} = -\pi_s^\theta(a)\pi_s^\theta(b), \tag{33}$$

- it satisfies the implicit constraints

$$0\leqslant\pi_s(a)\leqslant 1 \quad \forall s\in\mathcal{S}, a\in A(s), \tag{34}$$

$$\sum_{a\in A(s)} \pi_s(a) = 1 \quad \forall s\in\mathcal{S}. \tag{35}$$

In practice, we must impose that $-\theta_{max}\leqslant\theta_s(a)\leqslant\theta_{max}$ with θ_{max} being neither too small nor too large. On one hand, if θ_{max} is too large, the gradient ascent will slow down or even fail if the gradient components reach the machine precision limit. On the other hand, if θ_{max} is too small, the minimal entropy of the policy may become too large. Indeed, as θ_{max} decreases to zero, the policy space reduces progressively to maximum entropy policies, i.e. fully random policies.

If we define the entropy of a state s (with respect to a mixed policy π) as

$$-\sum_{a\in A(s)} \pi_s(a)\log\pi_s(a), \tag{36}$$

then it can be easily shown that the minimal state entropy is

$$\theta_{max}\frac{k_s e^{-\theta_{max}} - e^{\theta_{max}}}{k_s e^{-\theta_{max}} + e^{\theta_{max}}} + \log[k_s e^{-\theta_{max}} + e^{\theta_{max}}], \tag{37}$$

where $k_s = |A(s)| - 1$. Fig. 1 shows the evolution of the minimal state entropy for various values of $|A(s)|$. According to this figure and our experience, $3.0\leqslant\theta_{max}\leqslant 5.0$ is a good choice for Markov games with a small number of possible actions.

Note that it would also be possible to design an algorithm beginning with maximum entropy policies ($\theta_{max} = 0$) that progressively relaxes the entropy constraint by increasing (reasonably) θ_{max} in the same spirit as simulated annealing [40].

Now, if we furthermore replace every min operator by a softmin$^\phi$ operator, we obtain the *differentiable approximation*

$$\begin{aligned}\mathcal{L}^\phi_{\pi^\theta} = & \underset{o\in O(s_{init})}{\text{softmin}}\, \overline{Q}_{\pi^\theta}(s_{init},o) \\ & + \sum_{s\in\mathcal{S}\backslash\mathcal{D}} \sum_{a\in A(s)} \sum_{o\in O(s)} \lambda_{s,a,o}\Big[Q_{\pi^\theta}(s,a,o) - \Big[r_s(a,o) \\ & + \gamma \sum_{s_s\in\mathcal{S}} \left[T_{s,a,o,s_s} \underset{o_s\in O(s_s)}{\text{softmin}}\, \overline{Q}_{\pi^\theta}(s_s,o_s)\right]\Big]\Big] \\ & + \sum_{s\in\mathcal{D}} \sum_{a\in A(s)} \sum_{o\in O(s)} \lambda_{s,a,o}\left[Q_{\pi^\theta}(s,a,o) - 0\right]\end{aligned} \tag{38}$$

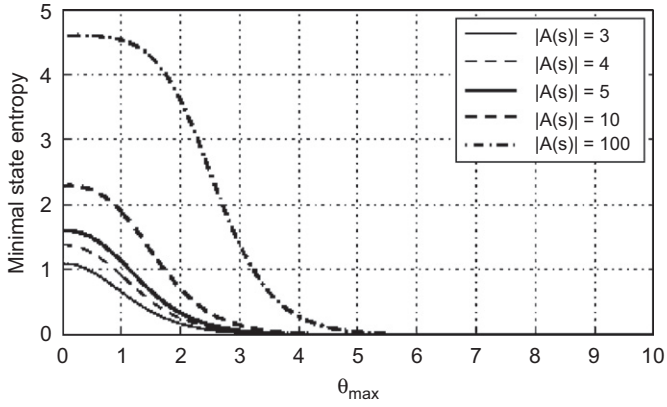

Fig. 1. Evolution of the minimal state entropy for $|A(s)| = 3, 4, 5, 10, 100$.

s.t. $\mathscr{L}_\pi = \lim_{\phi\to-\infty} \mathscr{L}_{\pi^\theta}^{\phi}$. Therefore, we obtain (see Appendix A for details) for each $s \in \mathscr{S}\backslash\{s_{init}\}, a \in A(s), o \in O(s)$

$$\lambda_{s,a,o} = \gamma\pi_s^\theta(a)\frac{\partial \text{softmin}_{o'\in O(s)}\,\overline{Q}_{\pi^\theta}(s,o')}{\partial\overline{Q}_{\pi^\theta}(s,o)} \times \sum_{s_p}\sum_{a_p\in A(s_p)}\sum_{o_p\in O(s_p)} T_{s_p,a_p,o_p,s}\lambda_{s_p,a_p,o_p} \tag{39}$$

and for each $a \in A(s_{init}), o \in O(s_{init})$

$$\lambda_{s_{init},a,o} = -\pi_s^\theta(a)\frac{\partial \text{softmin}_{o'\in O(s)}\,\overline{Q}_{\pi^\theta}(s,o')}{\partial\overline{Q}_{\pi^\theta}(s,o)} \tag{40}$$

and the gradient

$$\nabla_\theta \mathscr{L}_{\pi^\theta}^{\phi} = \begin{pmatrix} \vdots \\ \nabla_{\theta_{s,a}}\mathscr{L}_{\pi^\theta}^{\phi} \\ \vdots \end{pmatrix} \tag{41}$$

$$= \begin{pmatrix} \vdots \\ \sum_{o\in O(s)} \lambda_{s,a,o}[\overline{Q}_{\pi^\theta}(s,o) - Q_{\pi^\theta}(s,a,o)] \\ \vdots \end{pmatrix}. \tag{42}$$

Line-search cannot be used in our problem since, lacking T and r, we cannot evaluate the Lagrangian. We will rather use a *gradient-ascent scheme* inspired by optimisation theory and neural networks algorithms [18]: we follow the greatest-ascent direction:

$$\mathbf{d} = \begin{pmatrix} \vdots \\ d_s(a) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ \alpha_{\theta_s(a)}\nabla_{\theta_s(a)}\mathscr{L}_{\pi^\theta}^{\phi} \\ \vdots \end{pmatrix}, \tag{43}$$

where $\alpha_{\theta_s(a)}$ (each s, a pair has its own parameter) is updated after each iteration according to the rule

$$\alpha_{\theta_s(a)} \leftarrow \begin{cases} \alpha_{\theta_s(a)} * inc & \text{if } d_s(a)_t * d_s(a)_{t-1} > 0, \\ \alpha_{\theta_s(a)}/dec & \text{if } d_s(a)_t * d_s(a)_{t-1} < 0, \\ \alpha_{\theta_s(a)} & \text{if } d_s(a)_t * d_s(a)_{t-1} = 0, \end{cases} \tag{44}$$

where $inc, dec > 1$.

Since the Lagrange multipliers λ are defined by linear recursive relationships, we could theoretically compute their exact values by solving a linear system of equations. But our algorithm would not be able to tackle problems with huge or unknown state space. Yet, the Lagrange multipliers λ for a same state $s \in \mathscr{S}\backslash\mathscr{D}, s \neq s_{init}$ have most of their form in common (the shaded terms in the following equation):

$$\lambda_{s,a,o} = \gamma\,\pi_s^\theta(a)\frac{\partial \text{softmin}^{\phi}_{o'\in O(s)}\,\overline{Q}_{\pi^\theta}(s,o')}{\partial\overline{Q}_{\pi^\theta}(s,o)} \times \sum_{s_p}\sum_{a_p\in A(s_p)}\sum_{o_p\in O(s_p)} T_{s_p,a_p,o_p,s}\lambda_{s_p,a_p,o_p}. \tag{45}$$

If we accept to lose some efficiency, we can rather use a *local* gradient-ascent. Whereas the *global* gradient-ascent addresses the problem of updating every θ at the same time, the *local* version updates θ state by state. We obtain

$$\nabla_{\theta_s}\mathscr{L}_{\pi^\theta}^{\phi} \propto \begin{pmatrix} \vdots \\ -\pi_s^\theta(a)\sum_{o\in O(s)}\frac{\partial \text{softmin}_{o'\in O(s)}\,\overline{Q}_{\pi^\theta}(s,o')}{\partial\overline{Q}_{\pi^\theta}(s,o)}[\overline{Q}_{\pi^\theta}(s,o) - Q_{\pi^\theta}(s,a)] \\ \vdots \end{pmatrix} \tag{46}$$

and, eventually, the $\mathscr{2L}_2$ algorithm shown in Algorithm 5. At each epoch, the $\mathscr{2L}_2$ algorithm firstly updates $Q_\pi(s,a,o)$. Then, the successive components of the gradient are computed, the $\theta_s(a)$ values are updated to maximise $V_\pi(ss_{init})$ and eventually the $\alpha_{\theta_{s,a}}$ variables which regulate the gradient ascent are updated according Eq. (44).

Algorithm 5. Learning part of the $\mathscr{2L}_2$ algorithm.

1. $Q_\pi(s,a,o) \leftarrow (1-\alpha)Q_\pi(s,a,o) + \alpha[r + \gamma\,\text{softmin}^{\phi}{}_{o_s\in O(s_s)}\,\overline{Q}_\pi(s_s,o_s)]$
2.
3. $\beta \leftarrow \partial\,\text{softmin}^{\phi}{}_{o'\in O(s)}\,\overline{Q}_{\pi^\theta}(s,o')/\partial\overline{Q}_{\pi^\theta}(s,o)$
4. **for all** $a \in A(s)$ **do**
5. $\quad\theta_s(a) \leftarrow \theta_s(a) - \alpha_{\theta_{s,a}}\pi_s^\theta(a)\sum_{o\in O(s)}\beta[\overline{Q}_{\pi^\theta}(s,o) - Q_{\pi^\theta}(s,a,o)]$
6. $\quad$Update $\alpha_{\theta_{s,a}}$
7. **end for**

3.5.4. $\mathscr{2L}_2^+$, a simplified variant of $\mathscr{2L}_2$

As it is, Algorithm 5 can still be simplified when $\phi \to -\infty$ (see Appendix B). In this case, we obtain the simplified algorithm $\mathscr{2L}_2^+$ shown on Algorithm 6, very easy to implement. At each epoch $\langle s,a,o,r,s_s\rangle$, the Q value is updated and the subset $\mathscr{M}(s)$ of the opponent's actions corresponding to the minimum $\overline{Q}$ value is computed. Then, for each action $a \in A(s)$, the $\theta_s(a)$ parameter is updated using the simplified local gradient-ascent rule. Thus, $\mathscr{2L}_2^+$ uses a surprisingly simple and cheap update rule for the θ parameters. Moreover, whereas $\mathscr{2L}_2$ requires that the reward remains small because the $\overline{Q}_\pi$ values appear in expressions of the form $e^{\phi\overline{Q}_\pi(s,o)}$, $\mathscr{2L}_2^+$ works for any reward signal amplitude.

Algorithm 6. Learning part of the $\mathscr{2L}_2^+$ algorithm.

1. $Q_\pi(s,a,o) \leftarrow (1-\alpha)Q_\pi(s,a,o) + \alpha[r + \gamma\min_{o_s\in O(s_s)}\overline{Q}_\pi(s_s,o_s)]$
2.
3. $\mathscr{M}(s) \leftarrow \{o \in O(s) | \overline{Q}_{\pi^\theta}(s,o) = \min_{o'\in O(s)}\overline{Q}_{\pi^\theta}(s,o')\}$
4. **for all** $a \in A(s)$ **do**
5. $\quad\theta_s(a) \leftarrow \theta_s(a) - \alpha_{\theta_{s,a}}\pi_s^\theta(a)/|\mathscr{M}(s)|\sum_{o\in\mathscr{M}(s)}[\overline{Q}_{\pi^\theta}(s,o) - Q_{\pi^\theta}(s,a,o)]$
6. $\quad$Update $\alpha_{\theta_{s,a}}$
7. **end for**

3.5.5. Use of ϕ to smooth the Q values with $\mathscr{2L}_2$

In Section 3.5.2, we have mentioned that the softmin$^\phi$ operator tends to behave like the *mean* operator when $\phi \to 0$. It means that $\mathscr{2L}_2$ will smooth the differences between $\overline{Q}_\pi$ values in softmin$^\phi$

expressions if $\phi \to 0$. Yet, in the beginning of the learning, Q values may be very different from their actual values, thus it would make sense to use $\phi \approx 0$ to smooth these differences: this way, we would not favour any action when the Q values are still uncertain. Afterward, ϕ should be decreased during the learning, as the Q values converge, in order to relax the constraint on the policy computed by $2\mathscr{L}_2$. Note that $2\mathscr{L}_2^+$ does not offer this possibility, since every softmin$^\phi$ operator has disappeared in the simplification process.

4. Experiments

We will now assess the convergence and efficiency of the algorithms on two two-agent RL problems. Firstly, we will check that each algorithm behaves as expected and in particular that minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ all converge to optimal policies. Secondly, we will consider the convergence speed and the computation time needs for each algorithm. In particular, we will check that $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ converge in about the same numbers of epochs than minimax-Q but are faster in term of computational time.

In this section, we will consider simulated games. Each *experiment* consisted of several *runs* where several games, or *episodes*, were simulated with agents learning from each turn, or *epoch*.

4.1. The problems

In the finite two-player zero-sum Markov games used for our tests, the first and second players will be denoted, respectively, $\mathscr{A}g$ and $\mathscr{O}pp$. Except when otherwise stated, we will only mention $\mathscr{A}g$'s rewards, since the sum of the rewards is zero.

4.1.1. Rock Paper Scissors

Rock Paper Scissors ($\mathscr{RPS}_3$) is a simultaneous children's game which is perfect for a first insight because of its extreme simplicity. Each player must choose amongst *Rock*, *Paper* and *Scissors*, hence the name of the game, and both hand signals are simultaneously revealed by $\mathscr{A}g$ and $\mathscr{O}pp$ to determine the winner. The corresponding deterministic Markov game has thus only two states: s_{init} for the waiting of the hand signals and s_{out} for the outcome of the game (see Fig. 2).

The reward signal for the $\mathscr{RPS}_3$ problem is specified in s_{init} by the payoffs matrix Table 2 (the reward is zero in s_{out}). Since this matrix is antisymmetric, the game is fair and $V^*(s_{init}) = 0$. The optimal mixed strategy is

$$\left\langle \pi^*_{s_{init}}(\text{Rock}), \pi^*_{s_{init}}(\text{Paper}), \pi^*_{s_{init}}(\text{Scissors}) \right\rangle = \left\langle \tfrac{1}{5}, \tfrac{1}{5}, \tfrac{3}{5} \right\rangle. \quad (47)$$

Note that $\mathscr{RPS}_3$ is not the standard Rock Paper Scissors game where the $+3$ and -3 in Table 2 are replaced, respectively, by $+1$ and -1. These modifications are intended for preventing the random policy to be optimal.

4.1.2. Soccer

Soccer ($\mathscr{SC}$) is a simplified soccer game [21,22] made up of a simple horizontally oriented $m \times n$ grid ($n > 1$) with a goal of size m_g on both sides. The task for $\mathscr{A}g$ is to put the ball in $\mathscr{O}pp$'s goal (and *vice versa*). The reward is either $+1$ for a win, 0 for a draw and -1 for a loss; 0 otherwise. The moves are simultaneous, and there

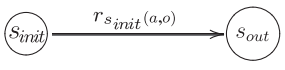

Fig. 2. Rock Paper Scissors state graph.

Table 2
Normal form of Rock Paper Scissors in s_{init}.

		$\mathscr{O}pp$		
		Rock	Paper	Scissors
$\mathscr{A}g$	Rock	0	-3	$+1$
	Paper	$+3$	0	-1
	Scissors	-1	$+1$	0

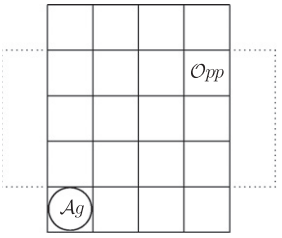

Fig. 3. A 5×4 Soccer environment with $m_g = 3$.

can be only one player by cell. When starting the game, the players are initially placed at random in the first and last columns, and the ball owner is randomly chosen (see Fig. 3). As a matter of fact, there are m^2 possible entry states and m^2 possible transitions from the initial fictitious state s_{init}. Fig. 3 shows an example of such an entry state for the 5×4 Soccer environment ($m_g = 3$), which is used for our tests, and for which $\mathscr{A}g$ owns the ball. If the ball owner moves to an already occupied cell, the ball goes to its opponent. Moreover, if both players try to move toward the same cell, the one which actually moves is chosen at random (and thus remains or becomes the ball owner). $\mathscr{SC}$ is therefore a stochastic game which allows checking how our algorithms behave in that case. Note that if the agents try to swap their places, nobody moves.

Every optimal policy for $\mathscr{SC}$ must be mixed. For example, consider the situation of Fig. 4. If π prescribes to go up, there is a possibility that $\mathscr{O}pp$ anticipates it and bars the way to $\mathscr{A}g$. On the contrary, if π prescribes equiprobably these two actions, any prediction by $\mathscr{O}pp$ becomes impossible and $\mathscr{A}g$ has a chance to create an opening. Note that the game has a probability 10^{-2} to end on a draw at each step in order to avoid deadlocks.

4.2. Experiments on $\mathscr{RPS}_3$

$\mathscr{RPS}_3$ is a deterministic Markov game, for which a mixed policy is necessary to achieve optimality. Its simplicity offers us an opportunity to perform a complete analysis for each algorithm: the optimal policy and their respective solution to $\mathscr{RPS}_3$ can be computed by hand. We will check that each algorithm behaves as expected and that minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ converge to optimal mixed policies.

4.2.1. Test protocol

The test protocol for the experiment on $\mathscr{RPS}_3$ consists of two steps: a learning phase and a testing phase.

Firstly, an instance of each algorithm is trained against the random agent $\mathscr{R}nd$. This learning phase lasts for 10^4 epochs in order to ensure complete convergence for every agent. Secondly, each agent is tested on one-to-one contests against $\mathscr{R}nd$ and a challenger in order to assess the quality of the learned policies.

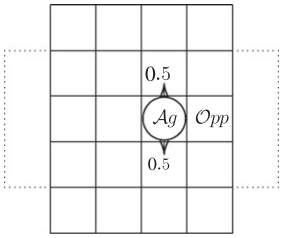

Fig. 4. A situation on a 5×4 Soccer environment with $m_g = 3$ where a mixed policy is necessary.

A *challenger* is a Q-learning agent, called $\mathscr{C}hg$, specifically trained against another agent which learning ability has been disabled. Challengers are expected to outperform Q-learning, whereas the performance of algorithms considering the rational player hypothesis should remain stable.

In order to analyse the result of the contests, we have measured for each game the *average effective return* $R_{\mathscr{O}pp}$, i.e. an estimator of the return in s_{init} that $\mathscr{A}g$ can expect to receive if it plays against $\mathscr{O}pp$.

In our experiments, we used the ε-greedy scheme with $\varepsilon = .9$ and the parameters of $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ are $dec = 1.1$, $inc = 1.01$, $\theta_{max} = 3$ and $\phi = -100$. Each contest consisted in 1000 episodes and each experiment was run 10 times. The $\alpha_{n_{i,s,a}}$ values for Q-learning were computed using $\alpha_{n_{i,s,a}} = k^i$ where $k = 10^{-3/100}$, i.e. $\alpha_{n_{i,s,a}}$ is decreased by 10^3 after 100 visits on the same state-action pair.

4.2.2. Preminilary analysis of the algorithms' behaviour

Q-learning solves two-agent RL problems by embedding the Markov game and its opponent in a MDP. The resulting policy takes advantage of the opponent's weaknesses, but may in turn be weak against a new opponent. Here, Q-learning will learn to beat specifically $\mathscr{R}nd$, therefore reducing Table 2 to Table 3(a). It will always choose Paper and thus $V_\pi(s_{init}) = R_{\mathscr{R}nd} = \frac{2}{3}$. However, $\mathscr{C}hg$(Q-learning) will thus reduce Table 2 to Table 3(b) and will always choose Scissors so that $R_{\mathscr{C}hg(\text{Q}-\text{learning})} = -1$.

Minimax, minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ rely on the rational player hypothesis. On one hand, Minimax will consider Table 4 and produce a pure policy prescribing either Paper or Scissors, so that $\mathscr{A}g$ will be certain to obtain at least $\text{maximin}_{\mathscr{RPS}_3} = -1$ and at most $\text{minimax}_{\mathscr{RPS}_3} = +1$. On the other hand, minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ will find the optimal mixed policy (see Eq. (47)). Thus, for these agents, $V_\pi(s_{init}) = 0$ and, moreover, $R_{\mathscr{O}pp} = 0$, whatever the opponent $\mathscr{O}pp$ is.

4.2.3. Results

Table 5 shows, for each algorithm, the mean and the 95% confidence interval on 10 runs of $V_\pi(s_{init})$, $R_{\mathscr{R}nd}$ and $R_{\mathscr{C}hg}$. As expected, for the one-agent algorithm, $R_{\mathscr{R}nd} \approx \frac{2}{3}$ whereas, for two-agent mixed algorithms, $R_{\mathscr{R}nd} \approx 0$: only the former has been able to take advantage of $\mathscr{R}nd$'s randomness, what the latter are not able of. Minimax obtains better results against $\mathscr{R}nd$ because its pure policy may prescribe either Paper or Scissors, which, respectively, yield 0 or $\frac{2}{3}$ against $\mathscr{R}nd$. On the contrary, if we compare the values of $R_{\mathscr{C}hg}$, Q-learning is totally outperformed by its challenger: $R_{\mathscr{C}hg} = -1$, whereas minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ still receive $R_{\mathscr{C}hg} \approx 0$. Minimax's return remains in $[-1, +1]$.

4.2.4. Discussion of the results

Each algorithm behaves according our expectations. Q-learning is able to take advantage of the opponent weaknesses, but is agent-dependent. If we change the opponent, the average effective return may decrease. The two-agent algorithms act the same whatever the opponent is, but they manage to secure their average effective return.

Moreover, since $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ perform as well as minimax-Q, we can expect that they converge to the true optimal policy. Indeed, their average policy on the 10 runs are, respectively,

$$\langle 0.199, 0.201, 0.600 \rangle \tag{48}$$

and

$$\langle 0.180, 0.200, 0.620 \rangle. \tag{49}$$

Note that the result for $2\mathscr{L}_2^+$ is perturbed because the convergence failed in one single run.

Table 3
Normal form of $\mathscr{RPS}_3$: (a) from the point of view of Q-learning learning against $\mathscr{R}nd$ and (b) from the challenger's point of view.

(a)

	$\mathscr{O}pp$ $\pi_{\mathscr{R}nd}$
Rock	$-\frac{2}{3}$
$\mathscr{A}g$ Paper	$+\frac{2}{3}$
Scissors	0

(b)

	$\mathscr{O}pp$		
	Rock	Paper	Scissors
$\mathscr{A}g$ Paper	+3	0	−1

Table 4
Maximin and minimax values for $\mathscr{RPS}_3$.

	$\mathscr{O}pp$ Rock	Paper	Scissors	min	
Rock	0	−3	+1	−3	
$\mathscr{A}g$ Paper	+3	0	−1	-1	← maximin
Scissors	−1	+1	0	-1	← maximin
max	+3	+1	+1		
		↑	↑		
		minimax			

4.3. Experiments on $\mathscr{SC}$

$\mathscr{SC}$ is a stochastic Markov game for which a mixed policy is necessary to achieve optimality. It offers us an opportunity to test the algorithms on a stochastic problem, more complex than $\mathscr{RPS}_3$ whose state graph is acyclic. We will check that $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ learn as fast as minimax-Q in terms of number of epochs, but faster in terms of computational time.

4.3.1. Test protocol

The test protocol for this experiment on a 5×4 soccer game with $m_g = 3$ (see Figs. 3 and 4) is inspired by Lagoudakis and Parr [21] and Littman [22].

Firstly, a minimax-Q agent, called mQ-ref, is trained against $\mathscr{R}nd$ for 10^7 epochs to be used as a reference for performance

Table 5
$V_\pi(s_{init})$, $R_{\mathscr{R}nd}$ and $R_{\mathscr{C}hg}$ obtained on $\mathscr{RPS}_3$ for different algorithms.

	Q-learning	Minimax	minimax-Q	$\mathscr{2L}_2$	$\mathscr{2L}_2^+$
$V_\pi(s_{init})$	0.712 [0.669 0.755]	−1	>-0.000	<0.000	−0.020 [−0.144 0.104]
$R_{\mathscr{R}nd}$	0.648 [0.638 0.658]	0.662 [0.628 0.695]	0.018 [0.008 0.027]	0.016 [0.008 0.024]	0.005 [−0.078 0.089]
$R_{\mathscr{C}hg}$	−1	−1	−0.009 [−0.016 −0.002]	−0.014 [−0.021 −0.006]	−0.027 [−0.158 0.104]

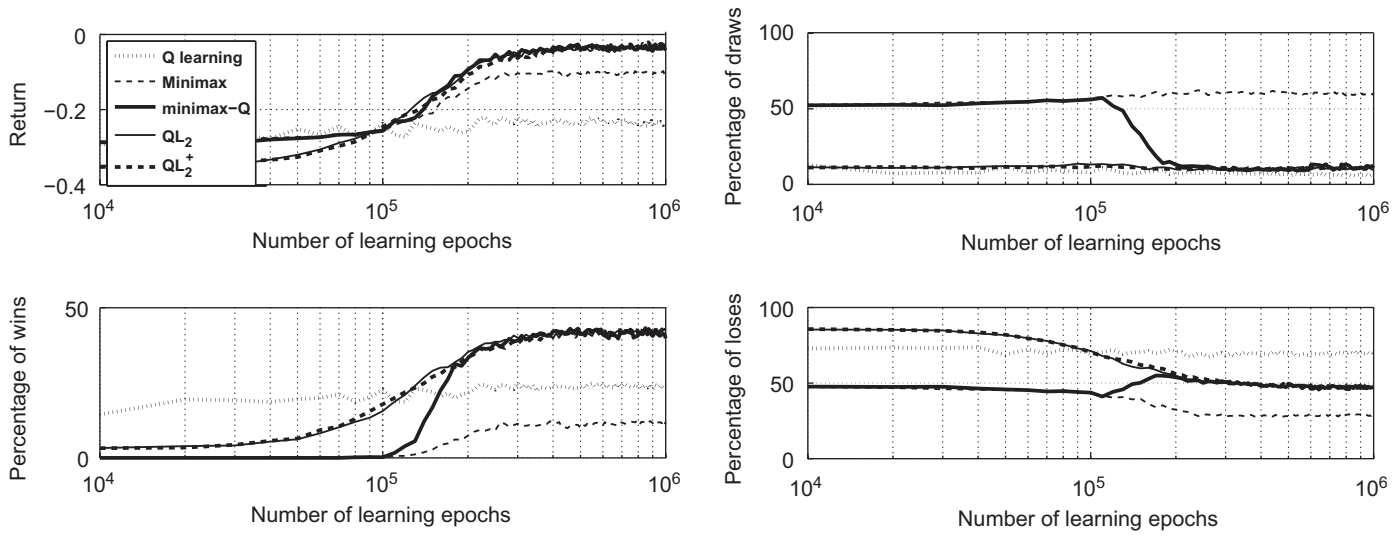

Fig. 5. $R_{\text{mQ-ref}}$ in s_{init} and percentage of wins, draws and losses for Q-learning, Minimax, minimax-Q, $\mathscr{2L}_2$ and $\mathscr{2L}_2^+$ learning on $\mathscr{SC}$ against $\mathscr{R}nd$ and tested against mQ-ref. Note the logarithmic scale.

Table 6
$R_{\text{mQ-ref}}$ in s_{init} and percentage of wins, draws and losses for Q-learning, Minimax, minimax-Q, $\mathscr{2L}_2$ and $\mathscr{2L}_2^+$ after learning on $\mathscr{SC}$ against $\mathscr{R}nd$.

	Q-learning	Minimax	minimax-Q	$\mathscr{2L}_2$	$\mathscr{2L}_2^+$
$R_{\text{mQ-ref}}$	−0.237 [−0.260 −0.213]	−0.103 [−0.112 −0.093]	−0.042 [−0.063 −0.020]	−0.030 [−0.040 −0.020]	−0.024 [−0.031 −0.017]
Wins	23.96 [21.75 26.17]	11.99 [9.39 14.59]	39.77 [35.62 43.92]	42.4 [41.05 43.75]	42.87 [41.24 44.50]
Draws	5.92 [5.42 6.42]	59 [55.14 62.86]	12.7 [6.45 18.95]	9.82 [8.96 10.68]	10.31 [8.14 12.49]
Losses	70.12 [67.92 72.32]	29.01 [27.61 30.41]	47.53 [44.40 50.66]	47.78 [46.59 48.97]	46.82 [45.83 47.81]

evaluation. Indeed, it has been shown in [34] that minimax-Q converges to optimal policies and it is therefore a good reference point to assess the learning of others agents. Secondly, two instances of each algorithm learn for 10^6 epochs by playing, respectively, against: (i) $\mathscr{R}nd$ and (ii) an agent simultaneously learning with the same algorithm. Every 10^4 epochs, each agent is tested in 1000 one-to-one contests against mQ-ref to estimate the percentage of wins, draws and losses and the average effective return $R_{\text{mQ-ref}}$ against the reference agent.

In our experiments, we used the ε-greedy scheme with $\varepsilon = .9$ and the parameters of $\mathscr{2L}_2$ and $\mathscr{2L}_2^+$ are $dec = 1.1$, $inc = 1.01$, $\theta_{max} = 3$ and $\phi = -100$. Each experiment was run 10 times. The $\alpha_{n_{i,s,a}}$ values were computed, for every algorithm, using $\alpha_{n_{i,s,a}} = k^n$ where $k = 10^{-3/10^6}$, i.e. $\alpha_{n_{i,s,a}}$ is decreased by 10^3 after the 10^6 visits on the same state-action pair.

4.3.2. Results against $\mathscr{R}nd$

Fig. 5 shows, for each algorithm learning against $\mathscr{R}nd$, the means on 10 runs of $R_{\text{mQ-ref}}$ and the percentage of wins, draws and losses of the first player against mQ-ref. Table 6 shows, for each algorithm, the mean and the 95% confidence interval on 10 runs of $R_{\text{mQ-ref}}$ and the percentage of wins, draws and losses after learning against $\mathscr{R}nd$.

Our results show that Q-learning is significantly worse than mQ-ref ($R_{\text{mQ-ref}} \approx -0.25$): most of the time (about 70%), it just loses. Since every optimal policy in $\mathscr{SC}$ is mixed, Minimax is also sub-optimal ($R_{\text{mQ-ref}} \approx -0.1$). However, it achieves an higher percentage of draws and a lower percentage of losses. Indeed, Minimax is more careful than Q-learning and does not consider that any new opponent will act as $\mathscr{R}nd$.

minimax-Q, $\mathscr{2L}_2$ and $\mathscr{2L}_2^+$ all converge to the optimal solution ($R_{\text{mQ-ref}} \approx 0$). When the game does not end on a draw (about 10%), they succeed in beating mQ-ref half the time. $\mathscr{2L}_2$ is a little below $R_{\text{mQ-ref}} = 0$, but not significantly.

4.3.3. Results against an agent simultaneously learning

Fig. 6 shows, for each algorithm against an agent simultaneously learning with the same algorithm, the means on 10 runs of $R_{\text{mQ-ref}}$ and the percentage of wins, draws and losses of the first player against mQ-ref. Table 7 shows, for each algorithm, the mean and the 95% confidence interval on 10 runs of $R_{\text{mQ-ref}}$ and

the percentage of wins, draws and losses after learning against an agent simultaneously learning with the same algorithm.

Q-learning is still sub-optimal ($R_{\text{mQ-ref}} \approx -0.1$), but it has significantly improved its behaviour. Indeed, $R_{\text{mQ-ref}}$, as well as the percentage of wins and losses, have been improved thanks to the learning opponent. In fact, it achieves performances similar to the those of Minimax which converges to the same values, whatever the opponent.

In opposition to Q-learning, the values obtained by minimax-Q, Minimax, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ after convergence have not changed, since they consider the rational player hypothesis. However, their convergence speed has been slightly improved.

4.3.4. Computational times comparison

During the experiment where the agents are learning against $\mathscr{R}nd$, we measured the total computational time used by each algorithm when both requiring an action and learning. Table 8 shows, for each algorithm, the mean and the 95% confidence interval on 10 runs of the computation time in μs used for one epoch. As expected, Q-learning and Minimax are faster: there is only a small difference between their computational times. $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ are much slower, but they still need about two times less computational time than minimax-Q.

4.3.5. Discussion of the results

The experiments on $\mathscr{SC}$ can be summarised in four points. Firstly, the one-agent algorithms heavily rely on the agent they are learning against to achieve good performances. They will most probably achieve optimality only in alternate games when the other agent is itself optimal (or becomes optimal during the learning), which is an unrealistic requirement. Secondly, Minimax is sub-optimal for simultaneous Markov games where mixed policies are necessary: whatever its learning opponent is, it will always achieve the same sub-optimal level, except if this latter prevents him to reach the states describing the task (here, the states right in front of the goal). Thirdly, minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ achieve optimality, whatever their opponent is, and converge at the same speed. Fourthly, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ need less computational time than minimax-Q.

4.4. Remarks on the use of $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$

Along the testing of $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ on $\mathscr{SC}$, we have noted that the choice of the θ_{max} value is important to achieve optimality. Indeed, if θ_{max} is too important, the gradient-ascent used in both algorithms is slow down, and the convergence speed becomes too small. This problem can be solved by decreasing θ_{max}, but when it becomes too small, the constraint on the policy may become too strict to achieve optimality.

5. Conclusion and further work

In this paper, we have used Markov games as a framework for RL, analysed algorithms for two-player zero-sum games and proposed the new algorithms $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$. Table 9 sum up the compared characteristics of Q-learning, Minimax, minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$.

5.1. One-agent algorithms

We have shown that, most of the time, Q-learning produces sub-optimal solutions because it embeds the opponent in the environment, which is therefore modelled as a MDP. Moreover,

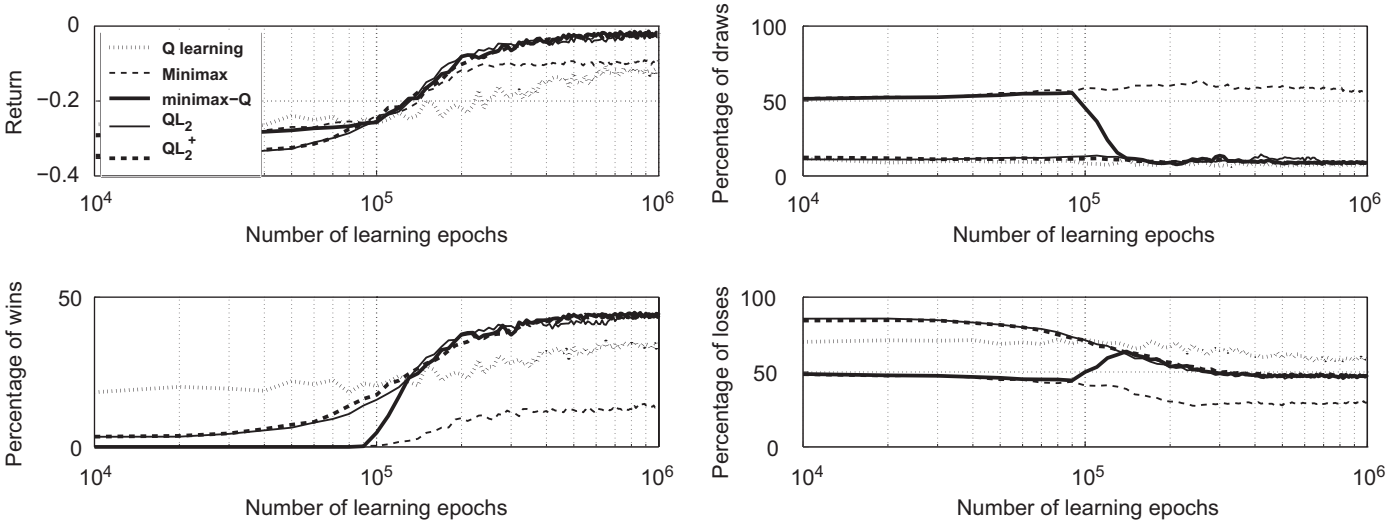

Fig. 6. $R_{\text{mQ-ref}}$ in s_{init} and percentage of wins, draws and losses for Q-learning, Minimax, minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ learning on $\mathscr{SC}$ against an agent simultaneously learning with the same algorithm and tested against mQ-ref.

Table 7
$R_{\text{mQ-ref}}$ in s_{init} and percentage of wins, draws and losses for Q-learning, Minimax, minimax-Q, $2\mathscr{L}_2$ and $2\mathscr{L}_2^+$ after learning on $\mathscr{SC}$ against an agent simultaneously learning with the same algorithm.

	Q-learning	Minimax	minimax-Q	$2\mathscr{L}_2$	$2\mathscr{L}_2^+$
$R_{\text{mQ-ref}}$	−0.118 [−0.141 −0.095]	−0.094 [−0.108 −0.080]	−0.017 [−0.025 −0.009]	−0.0274 [−0.036 −0.019]	−0.022 [−0.032 −0.013]
Wins	34.31 [32.42 36.20]	14.53 [11.16 17.90]	44.73 [43.86 45.60]	43.25 [42.63 43.87]	43.97 [42.88 45.06]
Draws	7.18 [6.07 8.29]	55.5 [51.12 59.88]	8.8 [8.24 9.36]	9.28 [8.41 10.15]	8.46 [8.07 8.85]
Losses	58.51 [55.93 61.09]	29.97 [28.60 31.34]	46.47 [45.33 47.61]	47.47 [46.72 48.22]	47.57 [46.37 48.77]

Table 8
Average computational time in μs used by different algorithms during one epoch when asked an action and when learning.

	Q-learning	Minimax	minimax-Q	$\mathcal{2L}_2$	$\mathcal{2L}_2^+$
Computational time	35	76	1680 [1674 1686]	905 [904 907]	729 [726 732]

Table 9
Heuristic Comparison of Q-learning, minimax, Minimax-Q, $\mathcal{2L}_2$ and $\mathcal{2L}_2^+$.

	Q-learning	Minimax	minimax-Q	$\mathcal{2L}_2$	$\mathcal{2L}_2^+$
Type of policy	Pure		Mixed		
Computation of the policy	Greedy action selection		Solving of a matrix game	Constrained optimisation of $V_\pi(s_{init})$	
Modelling of the opponent	Embedding in the environment	Rational player hypothesis			
Impact of the learning opponent	Helps the algorithms to find better policies	Improves the convergence speed			
Converge to an optimal policy?	Possibly in alternate games, depends on the learning opponent	Only when mixed policies are not necessary	Yes		

the embedding may fail if the opponent's behaviour is not Markovian or non-stationary.

The advantage of one-agent algorithms is that they do not consider the rational player hypothesis, which is useful when the problem is rather to beat a specific opponent. But they are opponent-dependent and may behave erratically if this latter's behaviour changes. Their performances increase greatly when they are learning against a good opponent ([22] made the same observation).

5.2. Two-agent algorithms

The latter category is composed of Minimax, minimax-Q and the two variant of our new algorithm, $\mathcal{2L}_2$ and $\mathcal{2L}_2^+$. Minimax is sufficient for alternate Markov games, but we have shown that pure policies may be sub-optimal for simultaneous Markov games, as predicted by game theory. Our results show that minimax-Q achieves optimality for such simultaneous games (see [22]), as well as $\mathcal{2L}_2$ and $\mathcal{2L}_2^+$. These three mixed algorithms converge in about the same number of epochs, but $\mathcal{2L}_2$ and $\mathcal{2L}_2^+$ appear to be faster in terms of computational time.

$\mathcal{2L}_2^+$ showed the best convergence speed amongst the three mixed algorithms in terms of computational time, but, as for $\mathcal{2L}_2$, the choice of θ_{max} is important to obtain convergence. Moreover, the convergence speed in terms of number of epochs can be improved by using a smart opponent for the learning, even if this latter has no impact on the final policies themselves.

The updating rule of $\mathcal{2L}_2^+$ consists of basic arithmetic operations and its only computational cost comes from the exponential involved in the policy evaluation.

5.3. Further work

In our opinion, there are still at least four issues in need of further work. Firstly, we should compare the algorithms on more complex problems: the problems tackled in this paper are toys problems whose complexities are far from those of real-world problems. Secondly, $\mathcal{2L}_2$ and $\mathcal{2L}_2^+$ can be improved: the gradient-ascent scheme and its robustness to high θ_{max} values could be enhanced. Moreover, the ϕ parameter has yet to be studied for $\mathcal{2L}_2$, as it could allows to smooth the Q values at the beginning of the learning. Thirdly, since the choice of θ_{max} constrains the minimal state entropy, we could increase the flexibility of the policies. For example, we could initially impose an high minimal state entropy and progressively relax the constraint during learning. Fourthly, it would be interesting to investigate the suitability of our constrained optimisation approach for n-players Markov games: (i) where cooperation is advantageous for both players [23] or (ii) where there are more than only two agents [35].

Acknowledgement

Thanks to Prof. F. Glineur (UCL, Belgium) for his ideas which helped us for the specific design of the simplex algorithm for minimax-Q.

Appendix A. Derivatives of the $\mathcal{2L}_2$ Lagrangian for stochastic environments

This appendix derives recursive relationships on the Lagrange multipliers for Eq. (38) and an analytical form for the gradient $\nabla_\theta \mathcal{L}^\phi_{\pi^\theta}$ for stochastic environments. Using π^θ and the softmin$^\phi$ operator, Eq. (38) defines $\mathcal{L}^\phi_{\pi^\theta}$ which can approximate as accurately as necessary $\mathcal{L}_\pi$, if we choose ϕ in consequence. In the next two sections, we will compute the derivatives of $\mathcal{L}^\phi_{\pi^\theta}$ with respect to the Q and θ values.

A.1. Recursive relationships on the Lagrange multipliers

For each $s \in \mathcal{S}\backslash\{s_{init}\}, a \in A(s), o \in O(s)$, the derivatives of $\mathcal{L}^\phi_{\pi^\theta}$ with respect to the Q values are

$$\frac{\partial \mathcal{L}^\phi_{\pi^\theta}}{\partial Q_{\pi^\theta}(s,a,o)} = \lambda_{s,a,o} - \gamma \frac{\partial \operatorname{softmin}^\phi_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial Q_{\pi^\theta}(s,a,o)} \times \sum_{s_p \in \mathcal{S}} \sum_{a_p \in A(s_p)} \sum_{o_p \in O(s_p)} T_{s_p,a_p,o_p,s} \lambda_{s_p,a_p,o_p} \tag{A.1}$$

and those for each $a \in A(s_{init}), o \in O(s_{init})$ are

$$\frac{\partial \mathcal{L}^\phi_{\pi^\theta}}{\partial Q_{\pi^\theta}(s_{init},a,o)} = \frac{\partial \operatorname{softmin}^\phi{}_{o' \in O(s_{init})} \overline{Q}_{\pi^\theta}(s_{init},o')}{\partial Q_{\pi^\theta}(s_{init},a,o)} + \lambda_{s_{init},a,o}. \tag{A.2}$$

If we use the chain rule, these derivatives become, respectively,

$$\frac{\partial \mathscr{L}^{\phi}_{\pi^\theta}}{\partial Q_{\pi^\theta}(s,a,o)} = \lambda_{s,a,o} - \gamma \pi^\theta_s(a) \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,a,o)} \times \sum_{s_p \in \mathscr{S}} \sum_{a_p \in A(s_p)} \sum_{o_p \in O(s_p)} T_{s_p,a_p,o_p,s} \lambda_{s_p,a_p,o_p} \tag{A.3}$$

and

$$\frac{\partial \mathscr{L}^{\phi}_{\pi^\theta}}{\partial Q_{\pi^\theta}(s_{init},a,o)} = \pi^\theta_s(a) \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s_{init})} \overline{Q}_{\pi^\theta}(s_{init},o')}{\partial \overline{Q}_{\pi^\theta}(s_{init},a,o)} + \lambda_{s_{init},a,o}. \tag{A.4}$$

Setting these quantities equal to zero for maximising $\mathscr{L}^{\phi}_{\pi^\theta}$, we obtain for each $s \in \mathscr{S} \backslash \{s_{init}\}, a \in A(s), o \in O(s)$ the recursive relationship on Lagrange multipliers λ

$$\lambda_{s,a,o} = \gamma \pi^\theta_s(a) \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} \times \sum_{s_p} \sum_{a_p \in A(s_p)} \sum_{o_p \in O(s_p)} T_{s_p,a_p,o_p,s} \lambda_{s_p,a_p,o_p} \tag{A.5}$$

and for each $a \in A(s_{init}), o \in O(s_{init})$ we have

$$\lambda_{s_{init},a,o} = -\pi^\theta_s(a) \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)}. \tag{A.6}$$

A.2. Analytical form for the gradient $\nabla_{\boldsymbol{\theta}} \mathscr{L}^{\boldsymbol{\phi}}_{\boldsymbol{\pi^\theta}}$

For each $s \in \mathscr{S}, a \in A(s)$, the derivative of $\mathscr{L}^{\phi}_{\pi^\theta}$ with respect to $\theta_s(a)$ is

$$\frac{\partial \mathscr{L}^{\phi}_{\pi^\theta}}{\partial \theta_s(a)} = -\gamma \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \pi^\theta_s(a)} \sum_{s_p \in \mathscr{S}} \sum_{a_p \in A(s_p)} \sum_{o_p \in O(s_p)} T_{s_p,a_p,o_p,s} \lambda_{s_p,a_p,o_p}. \tag{A.7}$$

We can develop the partial derivative term using the chain rule: we obtain successively

$$\sum_{o \in O(s)} \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} \frac{\partial \overline{Q}_{\pi^\theta}(s,o)}{\partial \pi^\theta_s(a)} \tag{A.8}$$

and

$$\sum_{o \in O(s)} \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} \left[\pi^\theta_s(a)(1-\pi^\theta_s(a)) Q_{\pi^\theta}(s,a,o) - \sum_{b \in A(s) \backslash \{a\}} \pi^\theta_s(a) \pi^\theta_s(b) Q_{\pi^\theta}(s,b,o) \right]. \tag{A.9}$$

Since

$$\overline{Q}_{\pi^\theta}(s,o) = \sum_{a \in A(s)} \pi^\theta_s(a) Q_{\pi^\theta}(s,a,o). \tag{A.10}$$

Eq. (A.9) becomes

$$\pi^\theta_s(a) \sum_{o \in O(s)} \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} [Q_{\pi^\theta}(s,a,o) - \overline{Q}_{\pi^\theta}(s,o)] \tag{A.11}$$

and, therefore, Eq. (A.7) becomes

$$\frac{\partial \mathscr{L}^{\phi}_{\pi^\theta}}{\partial \theta_s(a)} = -\gamma \pi^\theta_s(a) \left[\sum_{s_p \in \mathscr{S}} \sum_{a_p \in A(s_p)} \sum_{o_p \in O(s_p)} T_{s_p,a_p,o_p,s} \lambda_{s_p,a_p,o_p} \right] \times \left[\sum_{o \in O(s)} \frac{\partial \operatorname{softmin}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} [Q_{\pi^\theta}(s,a,o) - \overline{Q}_{\pi^\theta}(s,o)] \right]. \tag{A.12}$$

Using Eq. (A.6), we eventually obtain

$$\frac{\partial \mathscr{L}^{\phi}_{\pi^\theta}}{\partial \theta_s(a)} = \sum_{o \in O(s)} \lambda_{s,a,o} [\overline{Q}_{\pi^\theta}(s,o) - Q_{\pi^\theta}(s,a,o)]. \tag{A.13}$$

Appendix B. Derivation of the $2\mathscr{L}^{+}_{2}$ algorithm

In this appendix, we show how to simplify $2\mathscr{L}_2$ when $\phi \to -\infty$. Let us consider the term

$$\sum_{o \in O(s)} \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} [\overline{Q}_{\pi^\theta}(s,o) - Q_{\pi^\theta}(s,a,o)] \tag{B.1}$$

used in the instruction computing the update of the parameters $\theta_s(a)$ introduced in Eq. (31). The expression of the derivative is

$$\frac{e^{\phi \overline{Q}_{\pi^\theta}(s,o)} \left[1 + \phi (\overline{Q}_{\pi^\theta}(s,o) - \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')) \right]}{\sum_{o' \in O(s)} e^{\phi \overline{Q}_{\pi^\theta}(s,o')}} \tag{B.2}$$

and, if we define

$$\mathscr{M}(s) = \{o \in O(s) | \overline{Q}_{\pi^\theta}(s,o) = \min_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')\}, \tag{B.3}$$

we can distinguish two cases when $\phi \to -\infty$:

(1) If $o \in \mathscr{M}(s)$, then

$$\overline{Q}_{\pi^\theta}(s,o) - \operatorname*{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o') = 0 \tag{B.4}$$

and

$$\lim_{\phi \to -\infty} \frac{e^{\phi \overline{Q}_{\pi^\theta}(s,o)}}{\sum_{o' \in O(s)} e^{\phi \overline{Q}_{\pi^\theta}(s,o')}} = \frac{1}{|\mathscr{M}(s)|} \tag{B.5}$$

so that

$$\lim_{\phi \to -\infty} \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} = \frac{1}{|\mathscr{M}(s)|}. \tag{B.6}$$

(2) If $o \notin \mathscr{M}(s)$, then

$$\lim_{\phi \to -\infty} \frac{e^{\phi \overline{Q}_{\pi^\theta}(s,o)}}{\sum_{o' \in O(s)} e^{\phi \overline{Q}_{\pi^\theta}(s,o')}} = 0 \tag{B.7}$$

so that

$$\lim_{\phi \to -\infty} \frac{\partial \operatorname{softmin}^{\phi}_{o' \in O(s)} \overline{Q}_{\pi^\theta}(s,o')}{\partial \overline{Q}_{\pi^\theta}(s,o)} = 0. \tag{B.8}$$

Therefore, Eq. (B.1) eventually becomes

$$\frac{1}{|\mathscr{M}(s)|} \sum_{o \in \mathscr{M}(s)} [\overline{Q}_{\pi^\theta}(s,o) - Q_{\pi^\theta}(s,a,o)]. \tag{B.9}$$

References

[1] Qash: a Python generic framework for reinforcement learning in *n*-player Markov games ⟨http://www.dice.ucl.ac.be/mlg⟩.

[2] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, M. Saerens, Optimal tuning of continual online exploration in reinforcement learning, in: Proceedings of the 16th International Conference on Artificial Neural Networks (ICANN-2006), 2006.

[3] M.S. Bazaraa, J.J. Jarvis, H.D. Sherali, Linear Programming and Network Flows, Wiley, New York, 1990.

[4] D. Bertsekas, J. Tsitsiklis, Neuro-Dynamic Programming, Athena Scientific, 1996.

[5] D.P. Bertsekas, Dynamic Programming and Stochastic Control, Academic Press, New York, 1976.

[6] K. Binmore, Fun and Games: A Text on Game Theory, D.C. Heath, 1991.

[7] J.A. Boyan, A.W. Moore, Generalization in reinforcement learning: safely approximating the value function, in: Advances in Neural Information Processing Systems, vol. 7, MIT Press, Cambridge, MA, 1995.
[8] R.I. Brafman, M. Tennenholtz, R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning, Journal of Machine Learning Research 3 (2002) 213–231.
[9] J.S. Bridle, Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition, Neurocomputing: Algorithms, Architectures and Applications (1990) 227–236.
[10] J.S. Bridle, Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters, in: Advances in Neural Information Processing Systems, vol. 2, Morgan Kaufmann Publishers Inc., Los Altos, CA, 1990.
[11] R. Callan, Artificial Intelligence, Palgrave McMillan, 2003.
[12] R.H. Crites, A.G. Barto, Improving elevator performance using reinforcement learning, in: Advances in Neural Information Processing Systems, vol. 8, MIT Press, Cambridge, MA, 1996.
[13] E.A. Feinberg, A. Shwartz, Handbook of Markov Decision Processes: Methods and Applications, Springer, Berlin, 2001.
[14] J.A. Filar, O.J. Vrieze, Competitive Markov Decision Processes, Springer, Berlin, 1997.
[15] B. Frénay, M. Saerens, QL_2, a simple reinforcement learning scheme for two-player zero-sum markov games, in: Proceedings of the 16th European Symposium on Artificial Neural Network (ESANN-2008), 2008.
[16] L.J. Goldstein, D. Schneider, M.J. Siegel, Finite Mathematics and its Applications, Prentice-Hall, USA, 2006.
[17] M.E. Harmon, S.S. Harmon, Reinforcement learning: a tutorial, ⟨http://www.nbu.bg/cogs/events/2000/Readings/Petrov/rltutorial.pdf⟩.
[18] S. Haykin, Neural Networks: a Comprehensive Foundation, second ed, Prentice-Hall, Englewood Cliffs, NJ, 1998.
[19] L.P. Kaelbling, M.L. Littman, A.P. Moore, Reinforcement learning: a survey, Journal of Artificial Intelligence Research 4 (1996) 237–285.
[20] V. Klee, G.J. Minty, How good is the simplex algorithm?, in: O. Shisha (Ed.), Inequalities, Academic Press, New York, 1972, pp. 159–175.
[21] M.G. Lagoudakis, R. Parr, Value function approximation in zero-sum markov games, in: Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI-2002), 2002.
[22] M.L. Littman, Markov games as a framework for multi-agent reinforcement learning, in: Proceedings of the 11th International Conference on Machine Learning (ICML-94), 1994.
[23] M.L. Littman, Friend-or-foe q-learning in general-sum games, in: Proceedings of the 18th International Conference on Machine Learning (ICML-2001), 2001.
[24] T.M. Mitchell, Machine Learning, McGraw-Hill International Editions, New York, 1997.
[25] J. Moody, M. Saffell, Learning to trade via direct reinforcement, IEEE Transactions on Neural Networks 12 (2001) 875–889.
[26] G. Owen, Game Theory, Academic Press, New York, 1982.
[27] E. Rasmusen, Games and Information: An Introduction to Game Theory, third ed, Blackwell Publishers, Oxford, 2001.
[28] S. Russel, P. Norvig, Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, NJ, 2003.
[29] S. Singh, D. Bertsekas, Reinforcement learning for dynamic channel allocation in cellular telephone systems, in: Advances in Neural Information Processing Systems, vol. 9, MIT Press, Cambridge, MA, 1997.
[30] T. Soderstrom, Discrete-Time Stochastic Systems: Estimation and Control, Prentice-Hall PTR, Englewood Cliffs, NJ, 1994.
[31] J.C. Spall, Introduction to Stochastic Search and Optimization, Wiley-Interscience, New York, 2003.
[32] P.D. Straffin, Game Theory and Strategy, The Mathematical Association of America, 1996.
[33] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[34] C. Szepesvári, M.L. Littman, Generalized markov decision processes: dynamic-programming and reinforcement-learning algorithms, in: Proceedings of the 13th International Conference of Machine Learning (ICML-96), 1996.
[35] M. Tan, Multi-agent reinforcement learning: Independent vs. cooperative learning, in: Readings in Agents, Morgan Kaufmann, Los Altos, CA, 1997, pp. 487–494.
[36] G. Tesauro, Temporal difference learning and td-gammon, Communications of the ACM 38 (3) (1995) 58–68.
[37] H.C. Tijms, A First Course in Stochastic Models, Wiley, New York, 2003.
[38] M.J. Todd, A lower bound on the number of iterations of primal-dual interior-point methods for linear programming, in: Proceedings of the 15th Biennial Conference on Numerical Analysis at Dundee, Numerical Analysis 1993, Longman Press, New York, 1994.
[39] W. Uther, M. Veloso, Generalizing adversarial reinforcement learning, Technical Report, in: Proceedings of the AAAI Fall Symposium on Model Directed Autonomous Systems, 1997.
[40] P.J.M. van Laarhoven, E.H.L. Aarts, Simulated Annealing: Theory and Applications, Kluwer Academic Publishers, Dordrecht, 1988.
[41] J. von Neumann, O. Morgenstern, Theory of Games and Economic Behavior, Science, Wiley, New York, 1964.
[42] C. Watkins, Learning from delayed rewards, Ph.D. Thesis, King's College, Cambridge, UK, 1989.
[43] C. Watkins, P. Dayan, Q-learning, Machine Learning 8 (3–4) (1992) 279–292.
[44] M.H. Wright, The interior-point revolution in optimization: history, recent developments, and lasting consequences, Bulletin of the American Mathematical Society 42 (2005) 39–56.
[45] W. Zhang, T.G. Dietterich, A reinforcement learning approach to job-shop scheduling, in: Proceedings of the 19th International Joint Conference on Artificial Intellience (IJCAI-2005), 1995.

Benoît Frénay received the Engineer's degree from the Université Catholique de Louvain (UCL), Belgium, in 2007. He is now Ph.D. student at the UCL Machine Learning Group. His main research interests include machine learning, reinforcement learning, Markov models, dynamic programming, automated ECG analysis, data clustering and probability density estimation.

Marco Saerens, after graduation, joined the IRIDIA Laboratory (the Artifcial Intelligence Laboratory, Universite Libre de Bruxelles (ULB), Belgium) as a Research Assistant and received the Ph.D. degree in Engineering in 1991. While remaining a Part-time Researcher at IRIDIA, he then worked as a Senior Researcher in the R&D Department of various industries, mainly in the felds of Speech Recognition, Data Mining, and Artifcial Intelligence. In 2002, he joined the Universite Catholique de Louvain (UCL) as a Professor in Computing Science. His main research interests include artifcial intelligence, machine learning, data mining, pattern recognition, and speech/language processing. He is a Member of the IEEE.