Reinforcement learning in a Nutshell

Conference Paper · January 2007

Source: DBLP

CITATIONS
11

READS
195

4 authors, including:

Verena Heidrich-Meisner
Christian-Albrechts-Universität zu Kiel
17 PUBLICATIONS **249** CITATIONS
SEE PROFILE

Martin Lauer
Karlsruhe Institute of Technology
74 PUBLICATIONS **913** CITATIONS
SEE PROFILE

Martin Riedmiller
University of Freiburg
161 PUBLICATIONS **7,594** CITATIONS
SEE PROFILE

Reinforcement Learning in a Nutshell

V. Heidrich-Meisner[1], M. Lauer[2], C. Igel[1] and M. Riedmiller[2]

1- Institut für Neuroinformatik, Ruhr-Universität Bochum, Germany

2- Neuroinformatics Group, University of Osnabrück, Germany

Abstract. We provide a concise introduction to basic approaches to reinforcement learning from the machine learning perspective. The focus is on value function and policy gradient methods. Some selected recent trends are highlighted.

1 Introduction

Reinforcement learning (RL, [1, 2]) subsumes biological and technical concepts for solving an abstract class of problems that can be described as follows: An agent (e.g., an animal, a robot, or just a computer program) living in an environment is supposed to find an optimal behavioral strategy while perceiving only limited feedback from the environment. The agent receives (not necessarily complete) information on the current state of the environment, can take actions, which may change the state of the environment, and receives reward or punishment signals, which reflect how appropriate the agent's behavior has been in the past. This reward signal may be sparse, delayed, and noisy. The goal of RL is to find a policy that maximizes the long-term reward. Compared to supervised learning, where training data provide information about the correct behavior in particular situations, the RL problem is more general and thus more difficult, since learning has to be based on considerably less knowledge.

Reinforcement learning is rooted in the neuronal and behavioral sciences, and recent results in computational neuroscience try to bridge the gap between formal RL algorithms and biological substrate (e.g., see [3, 4] and references therein). However, due to space restrictions we focus on technical RL in the following. In the next section, we summarize the basic mathematical concepts. In section 3 we describe solution methods for RL problems, with an emphasis on value function and policy gradient algorithms. Finally we present selected extensions and trends in RL.

2 Basic concepts

The formal description of basic RL problems are Markov decision processes (MDPs), and most solution methods are rooted in Bellman's optimality equation. In the following, we introduce these concepts.

2.1 Markov decision process

Markov decision processes model time-discrete stochastic state-transition automata. An MDP $= (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ consist of a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$,

the expected (immediate) rewards $\mathcal{R}^a_{s,s'}$ received at the transition from state s to state s' by executing action a, and transition probabilities $\mathcal{P}$. The probability that in state s action a takes the agent to state s' is given by $\mathcal{P}^a_{s,s'}$. At every point in time t (we presume discrete time, for RL in continuous time see, e.g., [5]), the MDP is in some state s_t. An agent chooses an action $a_t \in \mathcal{A}$ which causes a transition from state s_t to some successor state s_{t+1} with probability $\mathcal{P}^{a_t}_{s_t,s_{t+1}}$. The agent receives a scalar reward (or punishment) $r_{t+1} \in \mathbb{R}$ for choosing action a_t in state s_t. The Markov property requires that the probabilities of arriving in a state s_{t+1} and receiving a reward r_{t+1} only depend on the state s_t and the action a_t. They are independent of previous states, actions, and rewards (i.e., independent of $s_{t'}$, $a_{t'}$, $r_{t'+1}$ for $t' < t$). This restriction can be relaxed, see section 4.

The agent that interacts with the MDP is modeled in terms of a *policy*. A deterministic policy $\pi : \mathcal{S} \to \mathcal{A}$ is a mapping from the set of states to the set of actions. Applying π means always selecting action $\pi(s)$ in state s. This is a special case of a stochastic policy, which specifies a probability distribution over $\mathcal{A}$ for each state s, where $\pi(s, a)$ denotes the probability to choose action a in state s.

2.2 Bellman equation

The goal of RL at some time $t = 0$ is to find a policy that maximizes the accumulated sum of rewards over time $R_t = \sum_{t=0}^{\infty} \gamma^t r_{t+1}$, where $\gamma \in [0, 1]$ is called the discount factor and determines how strongly immediate rewards are weighted compared to rewards in the future. A discount factor $\gamma < 1$ guarantees that the future discounted return R_t is always a finite number if the immediate reward is bounded. Since the state transition process is random the actually observed accumulated (discounted) reward $\sum_{t=0}^{\infty} \gamma^t r_{t+1}$ might be different from the expected return $\mathbb{E}[R_t|\pi, s_0]$ that the agent gets on average applying policy π starting from some initial state s_0. The return has the recursive property $\sum_{t=0}^{\infty} \gamma^t r_{t+1} = r_0 + \gamma \sum_{t=1}^{\infty} \gamma^t r_{t+1}$. Its expectation conditioned on the current state s and the policy π is called the value V^π of state s:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | \pi, s_0 = s, \right] = \sum_{a \in \mathcal{A}} \pi(s, a) \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{s,s'} \left(\mathcal{R}^a_{ss'} + \gamma V^\pi(s')\right) \quad (1)$$

This fixed point equation for V^π is known as *Bellman equation*. In addition to this definition of value functions by infinite sums of expected future rewards, it is also possible to define value functions based on the average future reward or on finite sums of future rewards (see, e.g., [1] for details).

The value functions induce a partial order on the set of policies, namely $\pi_1 \geq \pi_2$ if $V^{\pi_1}(s) \geq V^{\pi_2}(s)$ for all states $s \in \mathcal{S}$. Since we are interested in learning a policy that accumulates as much reward as possible we are interested in finding a policy π^* that outperforms all other policies (i.e., $\pi^* \geq \pi$ for all policies π). It has been shown that such an optimal policy always exists although

it need not be unique. All optimal policies of an MDP share the same value function V^* and include at least one deterministic policy.

3 Learning optimal policies

Reinforcement learning algorithms can be broadly classified into critic-only, actor-only, and actor-critic methods. Each class can be further divided into model-based and model-free algorithms, depending on whether the algorithm needs or learns explicitly transition probabilities and expected rewards for state-action pairs.

3.1 Critic-only: Learning based on value functions

Some of the most important learning algorithms in RL are critic-only methods, which are based on the idea to first find the optimal value function and then to derive an optimal policy from this value function. A selection of these approaches is described in the following.

Dynamic programming For a finite state space, the Bellman equation yields a finite set of $|\mathcal{S}|$ linear equations, which can be solved using standard methods. A more general and often more practical way is to solve these equations (up to certain accuracy) using dynamic programming. The right hand side of the Bellman equation (1) can be interpreted as a mathematical operator T^π that maps a value function to another value function. Thus V^π can be seen as the only fixed point of a contraction mapping T^π. Due to Banach's fixed point theorem we can find V^π by iteratively applying operator T^π to some initial value function V. This observation yields the basic idea to determine V^π by starting with an arbitrary value function V and repeatedly apply T^π to it.

If we are interested in finding the optimal value function V^*, we can modify the Bellman equation (1) by writing

$$V^*(s) = \max_{a \in \mathcal{A}} \left(\sum_{s' \in \mathcal{S}} \mathcal{P}^a_{s,s'} \left(\mathcal{R}^a_{ss'} + \gamma V^*(s') \right) \right) \quad . \tag{2}$$

Similar to (1) this equation can be seen as a fixed point equation for some operator T^* on V^* defined by the right hand side of (2). Again, it can be shown that T^* is a contraction mapping [1]. Hence, V^* is unique and it can be approximated applying the operator T^* repeatedly on some arbitrary initial value function. Once having found V^* we can derive an optimal (deterministic) policy π^* evaluating V^* and always choosing the action which leads to the successor state with the highest value $\pi^*(s) = \arg\max_{a \in \mathcal{A}} (\sum_{s' \in \mathcal{S}} (\mathcal{R}^a_{ss'} + \gamma \mathcal{P}^a_{s,s'} V^*(s')))$.

On the basis of these theoretical findings we can define a learning algorithm called *value iteration*, which determines the optimal value function by applying T^* repeatedly and builds an optimal policy using the previous equation. Figure 1 shows the optimal value function of a very simple MDP with 16 states. The

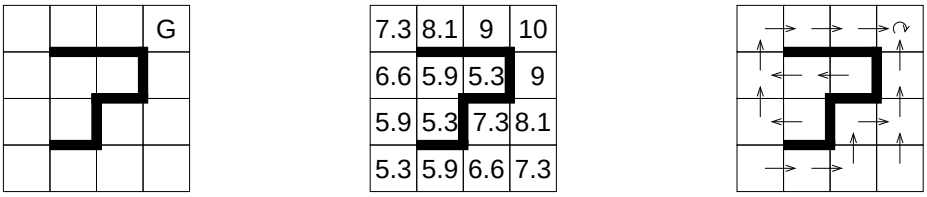

Fig. 1: Example of a simple maze world (left): a robot can move in a world of 16 states choosing actions for going up, down, left, right or stay in the current state. If it stays in the goal state (G) it will obtain a reward of 1, if it collides with a wall or tries to leave the grid world, it will get reward -1, and in all other cases reward 0. The robot starts in an arbitrary state. The figure in the middle shows the optimal value function V^* for this MDP with discount factor $\gamma = 0.9$. The figure on the right shows an optimal policy.

agent can derive an optimal policy by greedily selecting the action that takes the agent into the neighboring state with maximal expected reward.

Value iteration is a model-based approach, because it makes explicitly use of a model of the environment, which is given by the transition probabilities $\mathcal{P}$ and the expected rewards $\mathcal{R}$.

Temporal difference learning The main disadvantage of value iteration is the fact that the transition probabilities $\mathcal{P}$ and the expected rewards $\mathcal{R}$ must be known. In practice, the MDP is often unknown and the only way to get information about it is by interacting with the environment and observing rewards. Hence, we need an algorithm that estimates the value function and derives an optimal policy from a set of observable quantities: transitions from state s to state s' with action a and immediate reward r that was received. As a typical representative of model-free learning algorithms we consider Q-learning [6]. It learns an optimal policy by iteratively building an optimal value function. Defining a policy on top of a state value function V requires still some knowledge of the underlying process, since for all states s the possible successor states s' must be known. This is not necessary if we estimate the Q-function. For a fixed policy π, the Q-function models the expected accumulated reward if we play action a in state s and follow π afterwards:

$$Q^\pi(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{s,s'} \left(\mathcal{R}^a_{ss'} + \gamma V^\pi(s') \right) \tag{3}$$

Denoting with Q^* the Q-function of an optimal policy and comparing (3) and (2) we immediately get $V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$ and

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{s,s'} \left(\mathcal{R}^a_{ss'} + \max_{a' \in \mathcal{A}} Q^*(s',a') \right) \quad . \tag{4}$$

In the model-free setting, $\mathcal{P}$ and $\mathcal{R}$ are unknown so that we cannot use a value iteration like algorithm to find Q^*. Instead, we observe transitions of the

MDP represented by sets of quadruples (s, a, r, s') and approximate the optimal Q-function using a stochastic approximation approach [6]. From this idea, the *Q-learning* algorithm has been derived.

Q-learning iteratively builds a sequence of Q-functions. After observing a transition of the form $(s_t, a_t, r_{t+1}, s_{t+1})$ it performs an update of the Q-function at the observed state-action pair (s_t, a_t):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big(r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s_t, a_t)\big) \tag{5}$$

Here $\alpha \geq 0$ is a learning rate that is decreasing over time. This rule can also be interpreted as stochastic gradient descent with $r_{t+1} + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a') - Q(s_t, a_t)$ being the derivative of the Bellman error that measures the discrepancy between the left-hand side and right-hand side of (4). In the limit of an infinite set of observed transitions, convergence can be guaranteed if all state-action pairs occur infinitely often and some constraints on the decrease of the learning rate α over time are fulfilled (see, e.g., [1]). Once having calculated Q^* an optimal policy can be derived greedily by selecting the action a in state s that maximizes $Q^*(s, a)$.

Eligibility traces Although Q-learning and related approaches converge towards the optimal value function, the rate of convergence is low. One reason for this shortcoming is the fact that only a single state-action pair is updated per time step. To speed up learning the idea of *eligibility traces* has been introduced: the value function is not only updated for the previous state but also for the states which occurred earlier in the trajectory. Several learning algorithms have been developed on the basis of eligibility traces such as $Q(\lambda)$, SARSA(λ), and TD(λ), see [2] for an overview. We will focus on TD(λ) [7] in this paper.

TD(λ) is an algorithm to calculate the value function V^π for a given policy π. As in Q-learning, V^π is learned iteratively from observed trajectories $s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, r_3, \ldots$ of the state-transition system. In every step, the current error δ_t is calculated as $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, which can be interpreted as the temporal difference in the estimates of V^π concerning subsequent states. The goal of learning is to minimize these temporal difference errors. Algorithms based on this idea are therefore called temporal difference (TD) methods.

Instead of updating only $V(s_t)$ the value function is updated for all states. How strongly the current experience affects some state s depends on the time which has passed since s has occurred within the observed trajectory. This dependence is described in terms of an *eligibility* function $e(s)$. The update rule at a point in time t is

$$e(s) \leftarrow \begin{cases} \gamma\lambda e(s) + 1 & \text{if } s = s_t \\ \gamma\lambda e(s) & \text{if } s \neq s_t \end{cases} \quad \text{and} \quad V(s) \leftarrow V(s) + \alpha e(s)\delta_t \tag{6}$$

for all states $s \in \mathcal{S}$. The parameter $\lambda \in [0, 1]$ controls the rate at which the influence of an error decreases exponentially with the time difference between a

past state and the current state. In the case $\lambda = 0$ the value function is updated only for the current state s_t. It has been shown that the TD(λ) algorithm converges towards V^π with probability 1 if all states occur infinitely often (e.g., see[8]).

Approximate reinforcement learning The algorithms described above are based on calculating fixed points of value functions (V or Q). We assumed that we somehow can represent the value function in an appropriate way, e.g., by storing it in a table. However, in practice state spaces might become very large or even infinite so that a table based representation is not possible. Furthermore, filling these large tables would require a huge amount of observed transitions.

To overcome this problem, value functions are typically represented in terms of parametrized function approximators, e.g., linear functions, multi-layer perceptrons or grid based function approximators. Instead of updating individual entries of the value functions, the parameters of the function approximator are changed using gradient descent to minimize the Bellman error (e.g., $\sum_t \delta_t^2$, see [9]).

The convergence of these approaches cannot be guaranteed in general, only for linear approximators convergence has been proven under very restrictive conditions [10]. It can be shown that even small deviations from these conditions might lead to divergence [11].

For the case of a linear function approximators, learning a value function turns out to become a linear problem. Hence, instead of using gradient descent to minimize the Bellman error analytical solutions can be calculated [12, 13].

Building a world model So far *experience* of the agent implied interactions with its environment. But if the agent possesses an internal model of the environment, it can use this world model to create simulated experience and then learn from this second source of experience. The agent does not need to have knowledge about the MDP initially, but can estimate the model from its interaction with the environment. Thereby, model-based RL techniques can be applied without knowing $\mathcal{P}$ and $\mathcal{R}$ a priori. For example, Sutton's Dyna-Q [14] performs Q-learning using real and simulated experiences, where the real experience is simultaneously used to update the world model.

3.2 Actor-only: Direct policy search

Value function methods like TD-learning use an indirect approach, in the sense that instead of directly searching for an optimal policy, an optimal value function is learned before a policy is defined on top of it. In contrast, actor-only methods search directly in policy space. This is only possible if the search space is restricted. Typically a class of policies is parametrized by a real-valued parameter vector $\boldsymbol{\theta}$. The definition of this class allows to integrate prior knowledge about the task and thus reduce the search complexity. This can also be a drawback if the task is not known well enough to choose an appropriate subclass of policies.

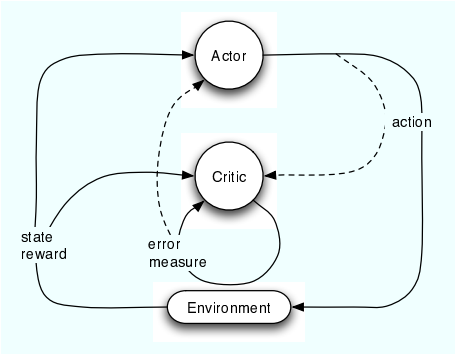

Fig. 2: General actor-critic architecture: Policy (actor) and value functions (critic) are represented and improved separately. The critic measures the performance of the actor and decides when the actor should be improved. This architecture offers the possibility to combine a value function approach (as the critic) with a policy gradient approach (as the actor).

An example for an actor-only method is William's REINFORCE algorithm [15] for immediate rewards, which is a special case of a policy gradient method, see section 3.3.

Evolutionary algorithms Evolutionary algorithms are randomized direct search algorithms inspired by concepts of neo-Darwinian evolution theory. Evolutionary methods can easily be adopted to solve RL tasks and turned out to be very robust in practice. Usually they are used to search directly in the space of policies [16]. The recent success of evolved NNs in game playing (e.g., [17, 18]) demonstrates the potential of evolutionary RL.

3.3 Actor-critic methods

Until now two types of algorithms have been introduced: *Actor-only* methods modifying policies directly and *critic-only* algorithms are based on evaluating value functions. Both approaches can be combined to *actor-critic* architectures, where the actor and critic are both represented explicitly and learned separately, see Fig. 2. The critic monitors the agents performance and determines when the policy should be changed. The actor-critic concept was originally introduced by Witten [19] and then by Barto, Sutton and Anderson [20], who coined the terms *actor* and *critic*. It has been particularly successful in neuronal RL. A detailed study of actor-critic algorithms is given in [21].

Policy gradient reinforcement learning Policy gradient strategies assume a differentiable structure on a predefined class of stochastic policies and ascent the gradient of a performance measure. The performance $\rho(\pi)$ of the current policy can be for example defined as $\rho(\pi) = \sum_{s,s' \in \mathcal{S}, a \in \mathcal{A}} d^\pi(s)\pi(s,a)\mathcal{P}^a_{s,s'}\mathcal{R}^a_{s,s'}$, where $d^\pi(s) = \lim_{t\to\infty} \Pr\{s_t = s \,|\, s_0, \pi\}$ is the stationary state distribution, which we assume to exist. The performance gradient $\nabla_{\boldsymbol{\theta}}\rho(\pi)$ with respect to the policy parameters $\boldsymbol{\theta}$ is estimated from interaction with the environment and the parameters $\boldsymbol{\theta}$ are adapted by gradient ascent along $\nabla_{\boldsymbol{\theta}}\rho(\pi)$. The policy gradient theorem (which is a result of a parametrized form of the Bellman equation) ensures that the performance gradient can be determined from unbiased estimates

of Q and d^π. For any MDP we have

$$\nabla_{\boldsymbol{\theta}} \rho = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi(s,a) Q^\pi(s,a) \ . \tag{7}$$

This formulation still contains explicitly the unknown value function $Q^\pi(s,a)$, which has to be estimated. It can be replaced by a function approximator $f_{\boldsymbol{w}} : S \times A \to \mathbb{R}$ (the *critic*) with real-valued parameter vector $\boldsymbol{w}$ satisfying the *convergence condition* $\sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi(s,a)(Q^\pi(s,a) - f_{\boldsymbol{w}}(s,a)) \nabla_{\boldsymbol{w}} f_{\boldsymbol{w}}(s,a) = 0$. This leads directly to the extension of the policy gradient theorem for function approximation [22]. If f_w satisfies the convergence condition and is compatible with the policy parametrization in the sense that $\nabla_w f_w(s,a) = \nabla_\theta \pi(s,a)/\pi(s,a)$ (i.e., f_w is linear in the corresponding features) then

$$\nabla_{\boldsymbol{\theta}} \rho(\pi) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \nabla_{\boldsymbol{\theta}} \pi(s,a) f_{\boldsymbol{w}}(s,a) \ . \tag{8}$$

Different policy gradient methods [22, 21, 23] vary in the way the performance gradient is estimated and the value function is approximated. Cao provides an unified view of these methods based on perturbation analysis [24].

Stochastic policies π with parameter $\boldsymbol{\theta}$ are parametrized probability distributions. In the space of probability distributions, the Fisher information matrix $F(\boldsymbol{\theta})$ induces an appropriate metric suggesting "natural" gradient ascent in the direction of $\tilde{\nabla}_{\boldsymbol{\theta}} \rho(\pi) = F(\boldsymbol{\theta})^{-1} \nabla_{\boldsymbol{\theta}} \rho(\pi)$. Using the definitions above, we have $F(\boldsymbol{\theta}) = \sum_{s \in \mathcal{S}} d^\pi(s) \sum_{a \in \mathcal{A}} \pi_{\boldsymbol{\theta}}(s,a) \nabla_{\boldsymbol{\theta}} \ln(\pi(s,a)) (\nabla_{\boldsymbol{\theta}} \ln(\pi(s,a)))^T$. This implies $\nabla_{\boldsymbol{\theta}} \rho = F(\boldsymbol{\theta}) \boldsymbol{w}$, which leads to the most interesting identity $\tilde{\nabla}_{\boldsymbol{\theta}} \rho(\pi) = \boldsymbol{w}$. Bagnell [25] builds an algorithm directly for the metric, while Peters et al. [26] integrate the natural gradient in an efficient actor-critic architecture.

3.4 Practical problems in reinforcement learning

Reinforcement learning algorithms have to deal with several inherent challenges, for example the *temporal credit assignment problem*, the *exploration-exploitation dilemma*, and the *curse of dimensionality*.

The reward the agent receives is not necessarily correlated to the last action the agent has performed, but can be delayed in time. The problem of mapping the reward only to those choices that led to it is called the *temporal credit assignment problem*.

In order to learn structure and features of an unknown environment, the agent needs to explore the state-action space. In the RL context, exploring means choosing actions that are regarded as suboptimal given the current state of knowledge. While exploration is necessary in the initial learning phase, it may become less relevant with increasing knowledge the agent has gathered by trial-and-error and that can be exploited for decision making. The E^3 algorithm and some Bayesian RL methods directly address the *exploration-exploitation dilemma* (see section 4).

The next problem is known as the *curse of dimensionality*. To ensure convergence a typical RL method needs to visit each state-action pair sufficiently often. Even comparatively simple toy problems can have huge state spaces that have to be explored. In many RL problems continuous state or action spaces are the appropriate description. Thus, testing all state-action pairs often becomes impractical. The agent has to generalize, that is, to learn from his experience about state-action pairs he never tried. Generalization can be achieved using function approximators, where gluing together neighboring states (e.g., through discretization) can be viewed as the most simple example. If a state is described by a vector of features, even for discrete representations the state space grows exponentially with the number of features. Achieving appropriate generalization requires a priori knowledge about the RL problem—and wrong generalization can severely harm the learning process.

4 Extensions and recent developments

In the following, we present selected extensions of the MDP formalism and recent trends in theory and practice of RL.

Partially observable Markov decision processes (POMDP) In a POMDP the agent does not necessarily receive full information about the state the system is in, but different states with different possibilities may look the same for the agent. A POMDP$(\mathcal{S}, y, \mathcal{Y}, \mathcal{A}, \mathcal{P}, \mathcal{R})$ contains additionally a set of observations $\mathcal{Y}$ and a function y that maps every state to an observation, which is provided to the agent instead of the true state information. Hence, the Markov property is violated from the agent's point of view.

It has been proven that finding an optimal policy in POMDPs is an NP-hard problem [27]. Learning algorithms based on value functions exist [28] but can only be applied to very small problems in practice. In order to distinguish between different states the agent perceives as identical it creates an internal model of what he believes to be the true state.

Stochastic games An important generalization of MDPs are stochastic games (or multi-agent MDPs) in which a team of agents interacts. Every agent may have its individual reward function, and optimality of a policy can only be interpreted as an optimal response to the behavior of the other agents. From the perspective of a single agent, the environment loses the property of being Markovian. Problems which occur in multi-agent environments are how to define the expected accumulated reward of an agent and how to coordinate all agents.

Some key ideas in multi-agent RL are drawn from game theory. For two agent games where both players have completely adversarial reward functions (zero-sum games), a learning algorithm similar to Q-learning has been proposed [29]. This concept has been generalized in [30]. Although the idea of Nash equilibria has brought some insights into multi-agent learning, the resulting approaches are

limited in practice to the cases of completely adversary or completely cooperative games [31].

Hierarchies Hierarchical models can be used to decompose RL problems. An overview is given in [32]. Several approaches have been developed, for example the concept of multi-step actions called *options* [33]. Options can either be normal actions or sequences of actions, where the latter solve particular subtasks. Using options, the MDP becomes a so-called semi-MDP. Semi-MDPs are also considered in the MAXQ approach [34], which decomposes the given MDP into a hierarchy of semi-MDPs whose solutions can be learned simultaneously. As a third representative of hierarchical RL, the HAM approach [35] restricts the choice of actions to a set of abstract machines which solve subtasks.

Bayesian reinforcement learning and kernels methods Bayesian and kernel methods have been very successful in a broad range of different fields in machine learning. A Bayesian approach to RL introduces a prior on a value function or a model (see [36, 37]). Experience is then used to compute a posterior over the value function or model, which captures all the knowledge the agent can have about the environment. On this basis, decisions that optimize future expected return yield theoretically an optimal trade-off between exploration and exploitation (see section 3.4).

Several kernel based methods have recently been developed in the context of RL (e.g., see [38]). These approaches use kernel methods as approximators for value functions or the unknown MDP and combine them with classical approaches such as dynamic programing.

E^3 (Explicit Explore or Exploit) algorithm The E^3 algorithm developed by Kaerns and Singh [39] uses an internal partial model of the underlying MDP, the known-state MDP. If S_{known} is the set of currently known states, the known-state MDP is the same as the full MDP for all states in S_{known}, but all transitions to states that are not elements of S_{known} are redirected to a single additional absorbing state. An optimal policy can either achieve its high performance by staying with high probability in S_{known} (exploitation case), or it leaves S_{known} with a significant probability (exploration case). The algorithm computes (off-line) an optimal exploitation and an optimal exploration policy, follows the exploitation policy if it achieves a predefined performance threshold, and follows the exploration policy otherwise. For a fixed number of steps, the E^3 algorithm achieves efficiently near-optimal return [39].

5 Conclusion

Reinforcement learning extends the domain of machine learning to a broad area of control and decision problems that cannot be tackled with supervised or unsupervised learning techniques. A couple of real-world applications have been successfully realized using RL, for example Tesauro's Backgammon player [40]

or applications to humanoid robot control (e.g., [26]) and helicopter control [41]. Some fundamental theoretical and practical problems still have to be solved in the domain of RL. However, promising recent developments have already led to considerably improved learning algorithms and new theoretical insights.

References

[1] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[2] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[3] S. Tanaka, K. Doya, G. Okada, K. Ueda, Y. Okamoto, and S. Yamawaki. Prediction of immediate and future rewards differentially recruits cortico-basal ganglia loops. *Nature Neuroscience*, 7(8):887–893, 2004.

[4] Y. Niv, M.O. Duff, and P. Dayan. Dopamine, uncertainty and TD learning. *Behavioral and Brain Functions*, 1(6), 2005.

[5] K. Doya. Reinforcement learning in continuous time and space. *Neural Computation*, 12:243–269, 2000.

[6] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

[7] R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[8] P. Dayan and T.J. Sejnowski. TD(λ) converges with probability 1. *Machine Learning*, 14(1):295–301, 1994.

[9] L.C. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proc. 12th Int'l Conf. on Machine Learning*, pages 30–37, 1995.

[10] J.N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 9, pages 1075–1081, 1997.

[11] A. Merke and R. Schoknecht. A necessary condition of convergence for reinforcement learning with function approximation. In *Proc. 19th Int'l Conf. on Machine Learning*, pages 411–418, 2002.

[12] S.J. Bradtke and A.G. Barto. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22(1–3):33–57, 1996.

[13] J.A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

[14] R.S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proc. 17th Int'l Conf. on Machine Learning*, pages 216–224, 1990.

[15] R.J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[16] D.E. Moriarty, A.C. Schultz, and J.J. Grefenstette. Evolutionary Algorithms for Reinforcement Learning. *Journal of Artificial Intelligence Research*, 11:199–229, 1999.

[17] K. Chellapilla and D.B. Fogel. Evolution, neural networks, games, and intelligence. *Proceedings of the IEEE*, 87(9):1471–1496, 1999.

[18] S.M. Lucas and G. Kendall. Evolutionary computation and games. *IEEE Computational Intelligence Magazine*, 1(1):10–18, 2006.

[19] I.H. Witten. An adaptive optimal controller for discrete-time markov environments. *Information and Control*, 34(4):286–295, 1977.

[20] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics*, 13:834–846, 1983.

[21] V.R. Konda and J.N. Tsitsiklis. On Actor-Critic Algorithms. *SIAM Journal on Control and Optimization*, 42(4):1143–1166, 2006.

[22] R.S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12, pages 1057–1063, 2000.

[23] J. Baxter and P.L. Bartlett. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15(4):319–350, 2001.

[24] X.R. Cao. A basic formula for online policy gradient algorithms. *IEEE Transactions on Automatic Control*, 50(5):696–699, 2005.

[25] J. Bagnell and J. Schneider. Covariant policy search. In *Proc. 18th Int'l Joint Conf. on Artificial Intelligence*, pages 1019–1024, 2003.

[26] J. Peters, S. Vijayakumar, and S. Schaal. Reinforcement learning for humanoid robotics. In *Proc. 3rd IEEE-RAS Int'l Conf. on Humanoid Robots*, pages 29–30, 2003.

[27] M.L. Littman. Memoryless policies: Theoretical limitations and practical results. In *From Animals to Animats 3: Proc. 3rd Int'l Conf. on Simulation of Adaptive Behavior*, Cambridge, MA, 1994.

[28] A.R. Cassandra, L.P. Kaelbling, and M.L. Littman. Acting optimally in partially observable stochastic domains. In *Proc. 12th National Conf. on Artificial Intelligence*, pages 1023–1028, 1994.

[29] M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proc. 11th Int'l Conf. on Machine Learning*, pages 157–163, 1994.

[30] J. Hu and M.P. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm. In *Proc. 15th Int'l Conf. on Machine Learning*, pages 242–250, 1998.

[31] M.L. Littman. Friend-or-foe Q-learning in general-sum games. In *Proc. 18th Int'l Conf. on Machine Learning*, pages 322–328, 2001.

[32] A.G. Barto and S. Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(4):341–379, 2003.

[33] R.S. Sutton, D. Precup, and S. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[34] T.G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[35] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems*, volume 10, pages 1043–1049, 1998.

[36] T. Wang, D. Lizotte, M. Bowling, and D. Schuurmans. Bayesian sparse sampling for online reward optimization. In *Proc. 22nd Int'l Conf. on Machine Learning*, pages 956–963, 2005.

[37] P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proc. 23rd Int'l Conf. on Machine Learning*, pages 697–704, 2006.

[38] Y. Engel, S. Mannor, and R. Meir. Reinforcement learning with Gaussian processes. In *Proc. 22nd Int'l Conf. on Machine learning*, pages 201–208. ACM Press New York, NY, USA, 2005.

[39] M. Kearns and S. Singh. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49(2):209–232, 2002.

[40] G. Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation*, 6(2):215–219, 1994.

[41] J. Bagnell and J. Schneider. Autonomous Helicopter Control using Reinforcement Learning Policy Search Methods. In *Proc. Int'l Conf. on Robotics and Automation*, 2001.