A Gentle Introduction to Reinforcement Learning

Ann Nowé(✉) and Tim Brys

Vrije Universiteit Brussel, Pleinlaan 2, Brussels, Belgium
{anowe,timbrys}@vub.ac.be

Abstract. This paper provides a gentle introduction to some of the basics of reinforcement learning, as well as pointers to more advanced topics within the field.

Keywords: Reinforcement learning · Heuristic information · Multi-objective · Multi-agent

1 Introduction

AI is quickly developing. Every year, the boundaries of what is possible are being pushed further and further. Since 2015, computers can play our video games from the 80's at a level comparable to that of an experienced gamer [20]. In 2016, they first beat the world champion of Go, the holy grail of board games, pulling moves that are inhuman, but "so beautiful" (*dixit Fan Hui*, reigning European champion). Fully self-driving cars have been around for at least a couple of years [15], and in a few more years Amazon drones will be whizzing around delivering packages to anybody and everybody [2].

One of the strengths of many of these systems is their ability to learn from data. The rules they follow, the behaviour they exhibit, is not exclusively programmed by some smart engineer. Rather, the engineer implements a learning algorithm, which is then fed data relevant for the task at hand. The learning algorithm then finds patterns in the data, discovers what are 'good' decisions for which situations, and an 'intelligent' system emerges.

This is *Machine Learning*.

2 Reinforcement Learning

Some of the examples cited above use a specific Machine Learning approach called reinforcement learning. This approach to learning is inspired by behaviourist psychology, where human and animal behaviour is studied from a reward and punishment perspective. A small illustrative example conveys the main principle of this learning theory:

Example 1. Say you want to train your dog to sit.
You take your dog outside and shout 'sit'.

S. Schockaert and P. Senellart (Eds.): SUM 2016, LNAI 9858, pp. 18–32, 2016.
DOI: 10.1007/978-3-319-45856-4_2

The dog realizes it needs to do something (you shouting and pointing to the ground is a definite clue), but it doesn't know what to do.
It barks, but nothing happens...
It gives a paw (something it learned before), but nothing happens...
It sits on the ground, and lo and behold, a dog cookie appears!

If you repeat this process many times, your dog will probably learn to associate the situation (you shouting 'sit') and its own action (sitting down), with the positive stimulus (a tasty cookie) and will repeat this behaviour on future occasions.

In essence, the learner is considered to crave 'something' that it receives depending on its behaviour; it receives more of it when it exhibits desirable behaviour, and less (or even something opposite) when it does not. Whether this 'something' be cookies for a dog, or dopamine in the human brain, or a simple numerical value, an increase of it tells the learner that it has done something right, and an intelligent learner will repeat that behaviour when it encounters a similar situation in the future.

This same principle was successfully used in the examples cited above to train AIs to play video games and play Go.[1] The former using the score in the game as reward, the latter the win or loss as reward or punishment.

3 The Reinforcement Learning Problem

In this section, we describe reinforcement learning (RL) more formally. We set the stage with the classic RL diagram, displayed in Fig. 1. It shows how an RL *agent* interacts with its *environment*. First, to say what an agent exactly is, is surprisingly difficult; definitions abound in AI literature. In this article, we adopt the following simple definition [25]:

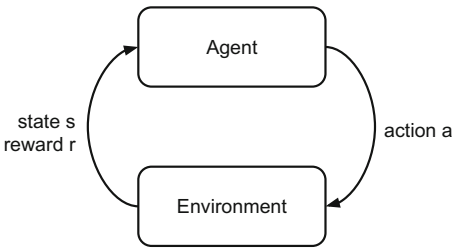

Fig. 1. The reinforcement learning agent-environment interaction loop.

[1] Do not dismiss these results as only academically interesting due to the 'game' nature of the problems: the complexity of these problems approaches and surpasses that of many more useful applications [28]. Furthermore, the past has shown that breakthrough advances in games have lead to breakthroughs in other fields. Monte Carlo Tree Search, initially developed for Go, is one example [8].

Definition 1 (An Agent.) *An agent is just something that perceives and acts.*

What the agent perceives and acts upon, we call the environment. This environment typically changes due to the agent's actions and possibly other factors outside the agent's influence. The agent perceives the state (s) of the environment (a potentially incomplete observation), and must decide which action (a) to take based on that information, such that the accumulation of rewards (r) it receives from the environment is maximized.

This agent-environment interaction process is most commonly formulated as a Markov Decision Process (MDP):

Definition 2 (Markov Decision Process). *A Markov Decision Process is a tuple* $\langle S, A, T, \gamma, R \rangle$.

- $S = \{s_1, s_2, \ldots\}$ *is the possibly infinite set of states the environment can be in.*
- $A = \{a_1, a_2, \ldots\}$ *is the possibly infinite set of actions the agent can take.*
- $T(s'|s, a)$ *defines the probability of ending up in environment state* s' *after taking action* a *in state* s.
- $\gamma \in [0, 1]$ *is the discount factor, which defines how important future rewards are.*
- $R(s, a, s')$ *is the possibly stochastic reward given for a state transition from* s *to* s' *through taking action* a. *It defines the goal of an agent interacting with the MDP, as it indicates the immediate quality of what the agent is doing.*

It is called a *Markov* Decision Process, since the state signal is assumed to have the Markov property:

Definition 3 (Markov Property). *A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it* [6].

In other words, the state signal should contain enough information to reliably predict future states.

The way an agent acts based on its perceptions, i.e., its behaviour, is commonly referred to as *a policy*, denoted as $\pi : S \times A \rightarrow [0, 1]$. It formally describes how likely an agent is to do something (action) in a given situation (state), by mapping state-action pairs to action selection probabilities. In this article, we use this notation for policies interchangeably with the following notation $\pi : S \rightarrow A$, which is a reformulation where not the action selection probabilities are output for a given state-action pair, but given a state and these probabilities, $\pi(s)$ outputs a probabilistically selected action.

The goal of an agent interacting with an MDP is to learn behaviour, a policy, that maximizes the discounted accumulation of rewards collected during its lifetime in the environment. This accumulation of reward up to a given time horizon or into infinity is called the return:

$$\mathcal{R}_t = R(s_t, a_t, s_{t+1}) + \gamma R(s_{t+1}, a_{t+1}, s_{t+2}) + \gamma^2 R(s_{t+2}, a_{t+2}, s_{t+3}) + \dots$$
$$= \sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1})$$

The discount factor γ determines the current value of future rewards. As $\gamma \to 1$, the agent becomes more farsighted, and will prefer large future rewards over smaller short-term rewards.

Given a state s and a policy π, we can express the return an agent can expect when starting from that state and following that policy as follows:

$$V^{\pi}(s) = E\left\{\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s\right\}$$

This *value function* expresses the quality of being in state s when following policy π, given the MDP to-be-solved that generates state transitions and rewards for these transitions. The expectation $E\{\}$ accounts for the stochasticity in these transition and reward functions, as well as in the policy that generates the action sequence.

Similarly, we can define the quality of being in state s, taking action a, and subsequently following policy π. This is called the action-value function:

$$Q^{\pi}(s, a) = E\left\{\sum_{k=0}^{\infty} \gamma^k R(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s, a_t = a\right\}$$

The expected returns encoded in these value functions yield a way to evaluate the quality of policies. A policy π is better than another policy π' if it has higher expected returns. A reinforcement learning agent needs to learn a policy that maximizes the expected return $\forall s \in S, a \in A$:

$$\pi^* = \arg\max_{\pi} Q^{\pi}(s, a)$$

π^* is called an optimal policy,[2] as it represents the behaviour that gets the highest return in expectation, thus solving the task encoded in the reward function.

4 Reinforcement Learning Algorithms

If the MDP's transition and reward functions are known, Dynamic Programming techniques can be used to optimally solve the problem [5]. Yet, it is uncommon to have a full specification of a system's dynamics or the reward function, and thus the use of techniques that can work with only knowledge of state and action spaces is necessary. These techniques must generate policies that maximize the expected return in environments with unknown dynamics and goals through trial-and-error. Learning a model of the environment may be part of

[2] There may be many, although their (action-)value functions will all be the same.

this process, but it is not necessary and many techniques are successful without this component.

As is the case in Dynamic Programming, there are basically two paradigms for reinforcement learning: policy iteration and value iteration. Both methods can be considered as on-line versions of their dynamic programming counterparts. In policy iteration, the learning agent contains two units, the evaluation unit and the action unit. The former is the internal evaluator, while the latter is responsible for determining the actions which look most promising according to the internal evaluator [4]. Policy gradient (PG) methods [34] are closely related to this policy iteration approach, with the internal evaluator usually replaced by sampled returns. These methods assume a parametric representation of the policy and the parameters are updated following a gradient in policy space. As they make assumptions on smoothness, these reinforcement learning techniques can naturally cope with continuous states and actions and uncertain state information. Exploration is typically achieved the same way as in policy iteration, i.e. by applying some noise on the action proposed by the current policy. Alternatively, exploration can also be realised by assuming a probability distribution (typically a Gaussian distribution) over the parameters involved in the policy. This approach is named Policy Gradients with Parameter-based Exploration (PGPE) [27]. Each time an action needs to be sampled, the parameters of the policy are drawn according to the distribution, resulting a policy instantiation that prescribes the action. The parameters of the policy are then updated based on the reward received using again a gradient approach. More advanced methods allow this idea to be applied to non-differential policies and also reduce the risk of getting stuck in a local optimum through a multi-modal approach [26].

In contrast, value iteration methods do not store a policy explicitly, but learn a value function from which they derive a policy. In the remainder of this section, we introduce and elaborate on temporal difference (TD) learning algorithms, a popular type of value iteration algorithms.

Definition 4 (TD Learning). *Temporal difference learning is an approach to reinforcement learning that keeps estimates of expected return and updates these estimates based on experiences and differences in estimates over successive time-steps.*

In other words, in TD learning, the agent incrementally updates estimates of a value function, using observed rewards and the previous estimates of that value function. One of the best known and simplest temporal difference learning algorithms is Q-learning [44]. It estimates the optimal Q-function Q^* by iteratively updating its estimates $\hat{Q}$ after each state-action-reward-next state (s, a, r, s') interaction with the environment:

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha\delta$$

$0 \leq \alpha \leq 1$ is the stepsize, controlling how much the value function is updated in the direction of the temporal difference error δ. The temporal difference error δ is the difference between the previous estimate and the observed sample:

$$\delta = r + \gamma \max_{a'} \hat{Q}(s', a') - \hat{Q}(s, a)$$

Q-learning performs *off-policy* learning. This means that it learns about a different policy than the one generating the interactions with the environment, which is called the behaviour policy. In the case of Q-learning, the policy being learned about is the optimal policy. The *on-policy* variant of Q-learning is called SARSA. It modifies the temporal difference error in such a way that the algorithm learns about the behaviour policy, using the action a' actually executed in next state s', instead of using the action with the highest estimate in that state:

$$\delta = r + \gamma \hat{Q}(s', a') - \hat{Q}(s, a)$$

If all state-action pairs are visited infinitely often, given some boundary conditions, Q-learning and SARSA are guaranteed to converge to the true Q-values [29,37]. In practice, a finite number of experiences is usually sufficient to generate near-optimal behaviour.

From an estimated Q-function, an agent can easily derive a greedy deterministic policy π:

$$\pi(s) = \arg\max_{a} \hat{Q}(s, a)$$

If the estimates have converged to the optimal Q-values Q^*, then this formula generates an optimal policy.

Since an agent typically needs to sufficiently explore the state-action space in order to find optimal behaviour (infinitely often in the case of Q-learning), it is in most cases insufficient to just use the greedy policy derived from the agent's estimates to generate interactions with the environment. That is because initial underestimation of the quality of actions might lead the agent to always select the first action it tries, because it yielded a higher return than expected and than estimated for the other actions. This results in the agent ceasing exploration prematurely, and the value function converging to a local optimum. Instead of always using the greedy policy with respect to the estimates to select actions, it is therefore often advisable to inject stochasticity into the policy to generate the necessary exploration. One way is to take a random action at every time-step with probability ϵ. This ensures that every reachable state-action pair has a non-zero visitation probability, irrespective of the estimated Q-values at that time. Let $\xi \in [0, 1]$ be a randomly drawn real number:

$$\pi(s) = \begin{cases} \text{a random action } \textit{if } \xi < \epsilon \\ \arg\max_a \hat{Q}(s, a) \ \textit{otherwise} \end{cases}$$

This is called ϵ-greedy action selection.

Another popular approach is softmax action selection, which determines the probability of every action based on the relative magnitude of the actions' estimates:

$$\pi(s, a) = \frac{e^{\frac{Q(s,a)}{\tau}}}{\sum_{a'} e^{\frac{Q(s,a')}{\tau}}}$$

The 'temperature' parameter τ determines how random (high τ) or greedy (low τ) action selection is. Actions with higher estimated Q-values will have relatively higher probabilities of being selected, and actions with lower estimated Q-values will have proportionally lower probabilities.

Defining a reward function requires some experience, however coming up with a reward function is often quite straightforward. Consider for example the case where we want an RL agent to find its way in unknown maze. Then we can give a reward of say $+100$ when it reaches its goal and 0 otherwise. Similarly, it we want the agent to learn to play chess, then we reward it with, e.g. $+100$ when it enters a winning state, -100 when it losses and 0 for all other states. Reward functions are not unique, consider for example the well known cart-pole problem. We can give the agent a reward of $+1$ at each time step it keeps the system under control, as such the agent will try to keep the system under control as long as possible, in order to collect as many $+1$'s as possible. Another way to express the same goals, is to reward the agent with a 0 and only when it fails, punishing it with a -1, combined with a discount factor γ strictly smaller than 1, this results in a reward of $-1 \times \gamma^t$, with t the time step of failure.

5 Function Approximation and Eligibility Traces

The basic versions of the algorithms described above are defined for discrete state-action spaces. They use a simple table to store the Q estimates: one entry for every possible state-action pair. Since many practical reinforcement learning problems have very large and/or continuous state spaces,[3] basic tabular learning methods are impractical, due to the sheer size of storage required, or even unusable, due to a table's inherent inability to faithfully represent continuous spaces. Therefore, function approximation techniques are required to render the learning problem tractable. Many different approximators exist, with deep neural networks being currently very much in vogue [20,28]. We introduce here a more basic and common function approximator, called tile-coding [1]. It is a linear approximator which overlays the state space with multiple randomly-offset, axis-parallel tilings. See Fig. 2 for an illustration. This allows for a discretization of the state-space, while the overlapping tilings guarantee a certain degree of generalization. The Q-function can be approximated by learning weights that map the tiles activated by the current state s and action a to an estimated Q-value:

$$\hat{Q}(s,a) = \theta^T \phi(s,a)$$

$\phi(s,a)$ is the feature vector representing state-action pair (s,a), i.e., a binary vector indicating the tiles activated by this state and the action, and θ is the parameter vector that needs to be learned to approximate the actual Q-function. This weight vector is updated using an update-rule similar to the one used in the tabular case:

$$\theta \leftarrow \theta + \alpha\delta$$

[3] Not to speak of continuous action spaces. That is not considered in this paper.

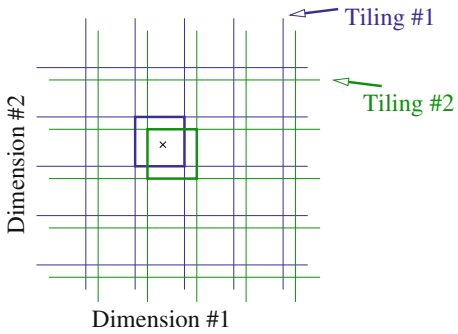

Fig. 2. An illustration of tile-coding function approximation on a two-dimensional state space, with two tilings. Figure taken from [35].

Alternatively, instead of discrete tilings, concepts from fuzzy set theory can be used, with fuzzy sets defined over the state space, and the membership operator $\mu(s)$ used as follows [14,22]:

$$\theta \leftarrow \theta + \alpha\mu(s)\delta$$

This update rule expresses that the more the state belongs to the fuzzy region described the fuzzy set, the more that sample is relevant for updating the Q-value associated to that region. Action selection techniques as described above can still be applied by combining the selected actions for each of the regions by weighting them according to the state membership.

A last commonly used mechanism in temporal difference reinforcement learning is called eligibility traces. Eligibility traces [17] are records of past occurrences of state-action pairs. These give a sense of how 'long ago' a given action was taken in a given state. They can be used to propagate reward further into the past (n-step) than the algorithms discussed until now do (one step). Using eligibility traces, not only the Q-value of the currently observed state-action pair is updated, but also those of past state-action pairs, inversely proportional to the time since they were experienced. Concretely, a (replacing) eligibility trace $e(s, a)$ for state s and action a is updated as follows [30]:

$$e(s,a) \leftarrow \begin{cases} 1 & s = s_t, a = a_t \\ \gamma\lambda e(s,a) & otherwise \end{cases}$$

It is set to 1 if (s, a) is currently observed, and otherwise it is decayed by $\gamma\lambda$, with $0 \leq \lambda \leq 1$ the eligibility trace decay parameter. Higher λ results in rewards being propagated further into the past. This eligibility trace update is performed at every step, thus making traces decay over time. The eligibility traces are included as a vector e in the Q update rule as follows:

$$\theta \leftarrow \theta + \alpha e\delta$$

6 Sample Complexity

As in general machine learning, *sample complexity* is important in reinforcement learning. Sample complexity represents the number of environment (s, a, s', r) samples an agent requires to perform a task well. Obtaining samples usually carries a cost, often greater than just the computational cost associated with processing the sample. Making a robot spend hours, days and weeks to learn a task is very costly. It takes a lot of electricity, several engineers to attend to the robot, and physical space for the robot to execute the task, none of which are cheap to obtain.

Therefore, one of the primary goals of reinforcement learning algorithms, besides convergence and (near-) optimality, is an efficient use of samples. The less samples an algorithm requires to achieve some desirable level of behaviour, the better. In general, there is an interplay between, setting of the learning rate, the value of the discount factor, the exploration strategy and the initialisation of e.g. the Q-values. Also the state description and the kind of function approximator plays a role in the learning performance. While the theoretical frame work for RL is the Markov decision processes, one can state that there is a graceful degradation, meaning the less Markovian the problem is (as perceived by the agent), the more carefully the exploration needs to be. This is especially the case in Multi-agent settings see Sect. 9.

Broadly speaking, researchers take either one of two approaches to reduce the number of samples an agent requires. They either build algorithms and techniques that inherently require less samples (one of the first algorithms of this kind was Dyna-Q [33][4], or they use some prior/external knowledge to bias the agent. Some argue that the former is superior to the latter, as it is the more general approach [32]. Yet, we believe that both will always be intertwined. One can see this for example in the success of AlphaGo [28], which definitely is a great example of new algorithms using their samples in a better way, yet still it required a great deal of human demonstrations to work well. One of the popular ways to include such external knowledge is through reward shaping.

7 Reward Shaping

The modern version of reward shaping, a technique with roots in behavioural psychology [31], provides a learning agent with extra intermediate rewards, much like a dog trainer would reward a dog for completing part of a task. This extra reward can enrich a sparse base reward signal (for example a signal that only gives a non-zero feedback when the agent reaches the goal), providing the agent with useful gradient information. This shaping reward F is added to the environment's reward R to create a new composite reward signal that the agent uses for learning:

[4] Dyna-Q combines Q-learning with learning a transition model. This (approximate) model is then used generated simulated samples for the Q-learner. Real life sample and simulated samples can be arbitrarily inter-twined. This principled is also referred to as planning in an RL context.

$$R_F(s,a,s') = R(s,a,s') + F(s,a,s')$$

Of course, since the reward function defines the task, modifying the reward function may modify the total order over policies, and make the agent converge to suboptimal policies (with respect to the environment's original reward).

If we define a potential function $\Phi : S \to \mathbb{R}$ over the state space, and take F as the difference between the new and old states' potential, Ng et al. proved that the total order over policies remains unchanged, and convergence guarantees are preserved [21]:

$$F(s,a,s') = \gamma\Phi(s') - \Phi(s) \tag{1}$$

Prior knowledge can be incorporated by defining the potential function Φ accordingly.

The definition of F and Φ was extended by [11,16,46] to include actions and timesteps, allowing for the incorporation of behavioural knowledge that reflects the quality of actions as well as states, and allowing the shaping to change over time:

$$F(s,a,t,s',a',t') = \gamma\Phi(s',a',t') - \Phi(s,a,t)$$

This extension also preserves the total order over policies and therefore does not change the task, given Ng's original assumptions. Harutyunyan et al. [16] use this result to show how any reward function $R^\dagger$ can be transformed into a potential-based shaping function, by learning a secondary Q-function $\Phi^\dagger$ in parallel on the negation of $R^\dagger$, and using that to perform dynamic shaping on the main reward R.

Many different types of knowledge can be used to bias a learning agent, ranging from expert knowledge [12] and human demonstrations [9] to knowledge transferred from a previous task [36] and on-line teacher advice [18].

How this different techniques relate to each other is discussed in the Ph.D. of Tim Brys[5]

8 Multi-objective Reinforcement Learning

Multi-objective reinforcement learning [24] (MORL) is a generalization of standard single-objective reinforcement learning, with the environment formulated as a multi-objective MDP, or MOMDP $\langle S, A, T, \gamma, \mathbf{R}\rangle$. The difference with the single-objective case is the reward function. Instead of returning a scalar value, it returns a vector of scalars, one for each of the m objectives:

$$\mathbf{R}(s,a,s') = [R_1(s,a,s'), \ldots, R_m(s,a,s')]$$

Policies are in this case evaluated by their expected vector returns $\mathbf{Q}^\pi$:

$$\begin{aligned}\mathbf{Q}^\pi(s,a) &= [Q_1^\pi(s,a), \ldots, Q_m^\pi(s,a)] \\ &= \left[E\left\{\textstyle\sum_{k=0}^{\infty}\gamma^k R_1(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s, a_t = a\right\}, \ldots,\right. \\ &\quad \left. E\left\{\textstyle\sum_{k=0}^{\infty}\gamma^k R_m(s_{t+k}, a_{t+k}, s_{t+k+1}) | s_t = s, a_t = a\right\}\right]\end{aligned}$$

[5] To appear, will be available online at ai.vub.ac.be.

Since there are multiple (possibly conflicting) signals to optimize, there is typically no total order over policies. Policies may be incomparable, i.e., the first is better on one objective while the second is better according to another objective, and thus the notion of optimality has to be redefined. A policy π_1 is said to strictly Pareto dominate another policy π_2, i.e., $\pi_1 \succ \pi_2$, if for each objective, π_1 performs at least as well as π_2, and it performs strictly better on at least one objective. The set of non-dominated policies is referred to as the *Pareto optimal set* or *Pareto front*. The goal in multi-objective reinforcement learning, and multi-objective optimization in general, is either to find a Pareto optimal solution, or to approximate the whole set of Pareto optimal solutions.

With a multi-objective variant of Q-learning, Q-values for each objective can be learned in parallel, stored as Q-vectors [13,41]:

$$\hat{\mathbf{Q}}(s,a) \leftarrow \hat{\mathbf{Q}}(s,a) + \alpha\boldsymbol{\delta}$$

$$\boldsymbol{\delta}_i = \mathbf{R}_i(s,a,s') + \gamma \max_{a'} \hat{\mathbf{Q}}_i(s',a') - \hat{\mathbf{Q}}_i(s,a)$$

The most common approach to derive a policy from these estimates is to calculate a linear scalarization, or weighted sum based on the estimated Q-vectors and a weight vector w [24,38,41]:

$$\pi(s) = \arg\max_a w^T \hat{\mathbf{Q}}(s,a)$$

The weight vector determines which trade-off solutions are preferred, although setting these weights *a priori* to achieve a particular trade-off is hard and non-intuitive [10], often requiring significant amounts of parameter tuning. Furthermore, because linear scalarization is a convex combination method, only solutions on convex parts of the Pareto-front can be found [39].

Algorithms that learn multiple trade-offs at the same time (multi-policy), and use operators that ensure access to both convex and concave parts of the Pareto-front are therefore very important. Only a restricted number of multi-policy MORL algorithms have been proposed so far. For instance, Barrett and Narayanan [3] propose the Convex Hull Value Iteration (CHVI) algorithm. From batch data, CHVI extracts and computes every linear combination of the objectives in order to obtain all deterministic optimal policies. As the algorithm relies on linear combinations, only policies on the convex hull are learned. The most computationally expensive operator is the procedure to compute and combine the convex hulls in the convex-hull version of the Bellman equation. Lizotte et al. [19] reduce the asymptotic space and time complexity of the bootstrapping rule by learning several value functions corresponding to different weight vectors using a piecewise linear spline representation. Wang and Sebag [43] propose a multi-objective Monte Carlo Tree Search (MO-MCTS) method to learn a set of solutions. The algorithm performs tree traversals by selecting the most promising actions. The upper confidence bounds of these actions are scalarized by applying the hypervolume indicator on the combination of their estimates and the set of Pareto optimal policies computed so far. Hence, a scalarized multi-objective value function is constructed that eases the process of selecting an action with

vectorial estimates. Finally, Pareto Q-Learning is to the best of our knowledge the only temporal-difference based multi-policy MORL algorithm [42]. It uses the Pareto dominance operator to selection actions, thus allowing for policies in concave areas of the Pareto front, and learns sets of Q-values by separately learning immediate rewards and expected future discounted rewards. It has been shown to be more sample efficient compared to for example MO-MCTS on a typical benchmark problem [40].

9 Multi-agent Reinforcement Learning

So far we have discussed approaches for single agent settings. However, when multiple learners simultaneously apply reinforcement learning in a shared environment, the traditional approaches often fail.

In a multi-agent setting, the assumptions that are needed to guarantee convergence, are often violated. Already in the most basic case where agents share a stationary environment and need to learn a strategy for a single interaction, many new complexities arise. These are mainly due to the fact that the agents are learning simultaneously and therefore the non-determinism in the reward signal might not only be due to the stochasticity of the environment, but also due to the actions taken by the other agents. Despite the added complexity, a real need for multi-agent systems exists. Often systems are inherently decentralized, and a central, single agent learning approach is not feasible because that would require too many resources or communication overhead. Examples of such systems are multi-robot set-ups, decentralized network routing, distributed load-balancing, electronic auctions, smart grids and traffic control. Depending on the characteristics of the system different multi-agent RL techniques might be more appropriate. The settings characteristics are for instance, whether the agents can observe each others actions or whether these actions are not observable by the other agents or only partially, whether the agents take their actions synchronously at fixed time steps or if they act asynchronously, whether the interactions are frequent or sparse, whether the rewards follow the actions instantaneously or are delayed (as for instance in queueing systems) and whether the agents have common or conflicting interests. In general, one can state that in a multi-agent setting, exploration is a very crucial aspect to make the reinforcement learning approach perform well. More precisely, exploration should be limited to allow the agent to differ some how between noise due to the environment and noise due the presence of other agents. Because of this, policy iteration techniques are interesting candidates in a Multi-agent Reinforcement Learning (MARL) setting, however value-iteration methods with specific exploration strategies have also been successfully applied. In case the agents have conflicting interests, the additional problem of the solution concept arises. As the agents have conflicting goals, it is no longer obvious what the solution of the system should be and where Game Theory becomes relevant. We refer the reader to [23] for a more in depth discussion on MARL. A recent paper by Bloembergen et al. [7], provides an overview of the dynamics of MORL techniques based on evolutionary game theory.

10 Conclusion

This paper gave a brief introduction to reinforcement learning basics, and some more recent extensions such as Multi-agent reinforcement learning and Multi-criteria reinforcement learning. We also gave some pointers to approaches to reduce the sample complexity, where reward shaping is a safe way to incorporated domain knowledge which recently received quite a lot of attention. We refer the reader to [33] for learning more about the basics of reinforcement learning and to [45] for an overview of some more advanced reinforcement learning algorithms.

References

1. Albus, J.S.: Brains, Behavior, and Robotics. Byte Books, Peterborough (1981)
2. Amazon: Amazon prime air (2016). http://www.amazon.com/b?node=8037720011. Accessed 20 Apr 2016
3. Barrett, L., Narayanan, S.: Learning all optimal policies with multiple criteria. In: Proceedings of the 25th International Conference on Machine Learning, pp. 41–47. ACM (2008)
4. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements that can solve difficult learning control problems. IEEE Trans. Syst. Man Cybern. **5**, 834–846 (1983)
5. Bertsekas, D.P.: Dynamic Programming and Optimal Control, vol. 1. Athena Scientific, Belmont (1995)
6. Bhattacharya, R., Waymire, E.C.: A Basic Course in Probability Theory. Springer, New York (2007)
7. Bloembergen, D., Tuyls, K., Hennes, D., Kaisers, M.: Evolutionary dynamics of multi-agent learning: a survey. J. Artif. Intell. Res. **53**, 659–697 (2015)
8. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**(1), 1–43 (2012)
9. Brys, T., Harutyunyan, A., Suay, H.B., Chernova, S., Taylor, M.E., Nowé, A.: Reinforcement learning from demonstration through shaping. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), pp. 3352–3358 (2015)
10. Das, I., Dennis, J.E.: A closer look at drawbacks of minimizing weighted sums of objectives for Pareto set generation in multicriteria optimization problems. Struct. Optim. **14**(1), 63–69 (1997)
11. Devlin, S., Kudenko, D.: Dynamic potential-based reward shaping. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems, vol. 1, pp. 433–440. International Foundation for Autonomous Agents and Multiagent Systems (2012)
12. Devlin, S., Kudenko, D., Grześ, M.: An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. Adv. Complex Syst. **14**(02), 251–278 (2011)
13. Gábor, Z., Kalmár, Z., Szepesvári, C.: Multi-criteria reinforcement learning. In: ICML, vol. 98, pp. 197–205 (1998)
14. Glorennec, P.Y.: Fuzzy q-learning and evolutionary strategy for adaptive fuzzy control. EUFIT **94**(1521), 35–40 (1994)

15. Google: Google self-driving car project. Accessed 20 Apr 2016
16. Harutyunyan, A., Devlin, S., Vrancx, P., Nowé, A.: Expressing arbitrary reward functions as potential-based advice. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (2015)
17. Klopf, A.H.: Brain function, adaptive systems: a heterostatic theory. Technical report AFCRL-72-0164, Air Force Cambridge Research Laboratories, Bedford, MA (1972)
18. Knox, W.B., Stone, P.: Combining manual feedback with subsequent MDP reward signals for reinforcement learning. In: Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems, pp. 5–12 (2010)
19. Lizotte, D.J., Bowling, M.H., Murphy, S.A.: Efficient reinforcement learning with multiple reward functions for randomized controlled trial analysis. In: Proceedings of the 27th International Conference on Machine Learning (ICML-2010), pp. 695–702 (2010)
20. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
21. Ng, A.Y., Harada, D., Russell, S.: Policy invariance under reward transformations: theory and application to reward shaping. In: Proceedings of the Sixteenth International Conference on Machine Learning, vol. 99, pp. 278–287 (1999)
22. Nowé, A.: Fuzzy reinforcement learning: an overview. In: Advances in Fuzzy Theory and Technology (1995)
23. Nowé, A., Vrancx, P., De Hauwere, Y.-M.: Game theory and multi-agent reinforcement learning. In: Wiering, M., van Otterlo, M. (eds.) Reinforcement Learning. ALO, vol. 12, pp. 441–470. Springer, Heidelberg (2012)
24. Roijers, D.M., Vamplew, P., Whiteson, S., Dazeley, R.: A survey of multi-objective sequential decision-making. J. Artif. Intell. Res. **48**, 67–113 (2013)
25. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Artificial Intelligence, vol. 25, p. 27. Prentice-Hall, Egnlewood Cliffs (1995)
26. Sehnke, F., Graves, A., Osendorfer, C., Schmidhuber, J.: Multimodal parameter-exploring policy gradients. In: Ninth International Conference on Machine Learning and Applications (ICMLA), pp. 113–118. IEEE (2010)
27. Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J.: Parameter-exploring policy gradients. Neural Netw. **23**(4), 551–559 (2010)
28. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
29. Singh, S., Jaakkola, T., Littman, M.L., Szepesvári, C.: Convergence results for single-step on-policy reinforcement-learning algorithms. Mach. Learn. **38**(3), 287–308 (2000)
30. Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Mach. Learn. **22**(1–3), 123–158 (1996)
31. Skinner, B.F.: The Behavior of Organisms: An Experimental Analysis. Appleton-Century, New York (1938)
32. Sutton, R.: The future of AI (2006). https://www.youtube.com/watch?v=pD-FWetbvN8. Accessed 28 June 2016
33. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction, vol. 1. Cambridge University Press, Cambridge (1998)

34. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y., et al.: Policy gradient methods for reinforcement learning with function approximation. In: NIPS, vol. 99, pp. 1057–1063 (1999)
35. Taylor, M.E.: Autonomous Inter-Task Transfer in Reinforcement Learning Domains. ProQuest, Ann Arbor (2008)
36. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: a survey. J. Mach. Learn. Res. **10**, 1633–1685 (2009)
37. Tsitsiklis, J.N.: Asynchronous stochastic approximation and Q-learning. Mach. Learn. **16**(3), 185–202 (1994)
38. Vamplew, P., Dazeley, R., Berry, A., Issabekov, R., Dekker, E.: Empirical evaluation methods for multiobjective reinforcement learning algorithms. Mach. Learn. **84**(1–2), 51–80 (2010)
39. Vamplew, P., Yearwood, J., Dazeley, R., Berry, A.: On the limitations of scalarisation for multi-objective reinforcement learning of Pareto fronts. In: Wobcke, W., Zhang, M. (eds.) AI 2008. LNCS (LNAI), vol. 5360, pp. 372–378. Springer, Heidelberg (2008)
40. Van Moffaert, K.: Multi-criteria reinforcement learning for sequential decision making problems. Ph.D. thesis, Vrije Universiteit Brussel (2016)
41. Van Moffaert, K., Drugan, M.M., Nowé, A.: Scalarized multi-objective reinforcement learning: novel design techniques. In: IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning. IEEE (2013)
42. Van Moffaert, K., Nowé, A.: Multi-objective reinforcement learning using sets of pareto dominating policies. J. Mach. Learn. Res. **15**(1), 3483–3512 (2014)
43. Wang, W., Sebag, M., et al.: Multi-objective monte-carlo tree search. In: ACML, pp. 507–522 (2012)
44. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, University of Cambridge (1989)
45. Wiering, M., Otterlo, M.: Reinforcement Learning: State-of-the-Art (Adaptation, Learning, and Optimization). Springer, Berlin (2012)
46. Wiewiora, E., Cottrell, G., Elkan, C.: Principled methods for advising reinforcement learning agents. In: International Conference on Machine Learning, pp. 792–799 (2003)

http://www.springer.com/978-3-319-45855-7

Scalable Uncertainty Management
10th International Conference, SUM 2016, Nice, France,
September 21-23, 2016, Proceedings
Schockaert, S.; Senellart, P. (Eds.)
2016, XI, 361 p. 64 illus., Softcover
ISBN: 978-3-319-45855-7