Technical report 10-028

Approximate dynamic programming and reinforcement learning*

L. Buşoniu, B. De Schutter, and R. Babuška

* This report can also be downloaded via https://pub.bartdeschutter.org/abs/10_028.html

Approximate dynamic programming and reinforcement learning

Lucian Buşoniu, Bart De Schutter, and Robert Babuška

Abstract Dynamic Programming (DP) and Reinforcement Learning (RL) can be used to address problems from a variety of fields, including automatic control, artificial intelligence, operations research, and economy. Many problems in these fields are described by continuous variables, whereas DP and RL can find exact solutions only in the discrete case. Therefore, approximation is essential in practical DP and RL. This chapter provides an in-depth review of the literature on approximate DP and RL in large or continuous-space, infinite-horizon problems. Value iteration, policy iteration, and policy search approaches are presented in turn. Model-based (DP) as well as online and batch model-free (RL) algorithms are discussed. We review theoretical guarantees on the approximate solutions produced by these algorithms. Numerical examples illustrate the behavior of several representative algorithms in practice. Techniques to automatically derive value function approximators are discussed, and a comparison between value iteration, policy iteration, and policy search is provided. The chapter closes with a discussion of open issues and promising research directions in approximate DP and RL.

Lucian Buşoniu
Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, 2628CD Delft, The Netherlands, e-mail: i.l.busoniu@tudelft.nl

Bart De Schutter
Delft Center for Systems and Control & Marine and Transport Technology Department, Delft University of Technology, Mekelweg 2, 2628CD Delft, The Netherlands, e-mail: b@deschutter.info

Robert Babuška
Delft Center for Systems and Control, Delft University of Technology, Mekelweg 2, 2628CD Delft, The Netherlands, e-mail: r.babuska@tudelft.nl

1 Introduction

Dynamic programming (DP) and reinforcement learning (RL) can be used to address important problems arising in a variety of fields, including e.g., automatic control, artificial intelligence, operations research, and economy. From the perspective of automatic control, the DP/RL framework comprises a nonlinear and stochastic optimal control problem [9]. Moreover, RL can be seen as adaptive optimal control [75, 83]. Algorithms that solve such a problem in general would be extremely useful for optimal control. From the perspective of artificial intelligence, RL promises a methodology to build an artificial agent that learns how to survive and optimize its behavior in an unknown environment, without requiring prior knowledge [74]. Developing such an agent is a central goal of artificial intelligence. Because of this mixed inheritance from optimal control and artificial intelligence, two sets of equivalent names and notations are used in DP and RL : e.g., 'controller' has the same meaning as 'agent', and 'process' has the same meaning as 'environment'. In this chapter, we will mainly use control-theoretical terminology and notations.

The DP/RL problem can be formalized as a Markov decision process (MDP) . In an MDP , at each discrete time step, the controller (agent) measures the state of the process (environment) and applies an action, according to a control (behavior) policy. As a result of this action, the process transits into a new state. A scalar reward is sent to the controller to indicate the quality of this transition. The controller measures the new state, and the whole cycle repeats. State transitions can generally be nonlinear and stochastic. This pattern of interaction is represented in Figure 1. The goal is to find a policy that maximizes the cumulative reward (the return) over the course of interaction [9, 11, 74].

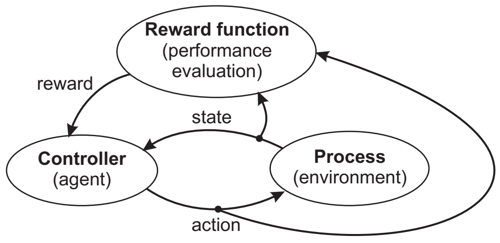


Fig. 1 The basic elements of DP and RL, and their flow of interaction.

As a conceptual example, consider a garbage-collecting robot. This robot measures its own position, and the positions of the surrounding objects; these positions are the state variables. The software of the robot is the controller (the agent). Note that in DP and RL , the process (environment) also includes the physical body of the robot. The controller receives the position measurements, and sends motion commands to the motors; these commands are the actions. The dynamics describe the rule according to which positions (states) change as a result of the commands (actions). The reward signal can, e.g., be positive at every time step in which the robot picks up trash, and zero otherwise. In this case, the goal is to pick up as much trash as possible, because this corresponds to accumulating as much reward as possible.

DP algorithms are model-based: they require a model of the MDP , in the form of the transition dynamics and the reward function [9, 11]. DP algorithms typically work offline, producing a policy which is then used to control the process. Usually, analytical expressions for the dynamics and the reward function are not required. Instead, given a state and an action, the model is only required to generate a next state and the corresponding reward. RL algorithms are model-free [74], and use transition and reward data obtained from the process. RL is useful when a model is difficult or costly to derive. So, RL can be seen as model-free, sample-based or trajectory-based DP , and DP can be seen as model-based RL . Some RL algorithms work offline, using data collected in advance. Other RL algorithms work online: they compute a solution while simultaneously controlling the process. Online RL is useful when it is difficult or costly to obtain data in advance. Online RL algorithms must balance the need to collect informative data with the need to control the process well.

DP and RL algorithms can be classified by the path they take to search for an optimal policy. *Value iteration* algorithms search for the optimal value function, i.e., the maximal returns as a function of the state and possibly of the control action. The optimal value function is then used to compute an optimal policy. *Policy iteration* algorithms iteratively improve policies. In each iteration, the value function of the current policy is found (instead of the optimal value function), and this value function is used to computed a new, improved policy. *Policy search* algorithms use optimization techniques to directly search for an optimal policy.[1]

Classical DP and RL algorithms require exact representations of the value functions and policies. When some of the variables have a very large or infinite number of possible values (e.g., when they are continuous), exact representations are no longer possible. Instead, value functions and policies need to be approximated. Since many problems of practical interest have large or continuous state and action spaces, approximation is essential in DP and RL . Two main types of approximators can be identified: parametric and nonparametric approximators. *Parametric* approximators are functions of a set of parameters; the form of the function is given a priori, and does not depend on the data. The parameters are tuned using data about the target value function or policy. A representative example is a linear combination of a fixed set of basis functions (BFs) . In contrast, the form and number of parameters of a *nonparametric* approximator are derived from the available data. For instance, kernel-based approximators can also be seen as representing the target function with a linear combination of BF s, but, unlike parametric approximation, they define one BF for each data point.

This chapter provides an in-depth review of the literature on approximate DP and RL in large or continuous-space, infinite-horizon problems. Approximate value iteration, policy iteration, and policy search are presented in detail and compared.

[1] A fourth category of (model-based) algorithms is *model predictive control* [8, 22], which we do not discuss in this chapter.

Model-based (DP), as well as online and batch model-free (RL) algorithms are discussed. Algorithm descriptions are complemented by theoretical guarantees on their performance, and by numerical examples illustrating their behavior in practice. We focus mainly on parametric approximation, but also mention some important nonparametric approaches. Whenever possible, we discuss the general case of nonlinearly parameterized approximators. Sometimes, we delve in more detail about linearly parameterized approximators, e.g., because they allow to derive better theoretical guarantees on the resulting approximate solutions.

The remainder of this chapter is structured as follows. After a brief introduction to classical, exact DP and RL in Section 2, the need for approximation in DP and RL is explained in Section 3. This is followed by an in-depth discussion of approximate value iteration in Section 4 and of approximate policy iteration in Section 5. Techniques to automatically derive value function approximators are reviewed in Section 6. Approximate policy search is discussed in Section 7. A representative algorithm from each class (value iteration, policy iteration, and policy search) is applied to an example involving the optimal control of a DC motor: respectively, grid Q-iteration in Section 4, least-squares policy iteration in Section 5, and pattern search policy optimization in Section 7. While not all of the algorithms used in the examples are taken directly from the literature, and some of them are designed by the authors, they are all straightforward instantiations of the class of techniques they represent. Approximate value iteration, policy iteration, and policy search are compared in Section 8. Section 9 closes the chapter with a discussion of open issues and promising research directions in approximate DP and RL .

2 Markov decision processes. Exact dynamic programming and reinforcement learning

This section formally describes MDP s and characterizes their optimal solution. Then, exact algorithms for value iteration and policy iteration are described. Because policy search is not typically used in exact DP and RL , it is not described in this section; instead, it will be presented in the context of approximation, in Section 7.

2.1 Markov decision processes and their solution

A Markov decision process (MDP) is defined by its state space X, its action space U, its transition probability function $\tilde{f} : X \times U \times X \to [0, \infty)$, which describes how the state changes as a result of the actions, and its reward function $\tilde{\rho} : X \times U \times X \to \mathbb{R}$, which evaluates the quality of state transitions. The controller behaves according to its control policy $h : X \to U$.

More formally, at each discrete time step k, given the state $x_k \in X$, the controller takes an action $u_k \in U$ according to the policy h. The probability that the resulting next state x_{k+1} belongs to a region $X_{k+1} \subset X$ of the state space is $\int_{X_{k+1}} \tilde{f}(x_k, u_k, x')\mathrm{d}x'$. For any x and u, $\tilde{f}(x, u, \cdot)$ is assumed to define a valid probability density of the argument '$\cdot$'. After the transition to x_{k+1}, a reward r_{k+1} is provided according to the reward function: $r_{k+1} = \tilde{\rho}(x_k, u_k, x_{k+1})$. The reward evaluates the immediate effect of action u_k, namely the transition from x_k to x_{k+1}. We assume that $\|\tilde{\rho}\|_\infty = \sup_{x,u,x'} |\tilde{\rho}(x, u, x')|$ is finite.[2] Given $\tilde{f}$ and $\tilde{\rho}$, the current state x_k and action u_k are sufficient to determine the probability density of the next state x_{k+1} and of the reward r_{k+1}. This is the so-called Markov property, which is essential in providing theoretical guarantees about DP/RL algorithms.

Note that, when the state space is countable (e.g., discrete), the transition function can also be given as $\bar{f} : X \times U \times X \to [0, 1]$, where $\bar{f}(x_k, u_k, x')$ is the probability of reaching x' after taking u_k in x_k. The function $\tilde{f}$ is a generalization of $\bar{f}$ to uncountable (e.g., continuous) state spaces; in such spaces, the probability of reaching a given point x' in the state space is generally 0, making a description of the form $\bar{f}$ inappropriate. Additionally, the individual rewards themselves can be stochastic; if they are, to simplify the notation, we take $\tilde{\rho}$ equal to the *expected* rewards.

Developing an analytical expression for the transition probability function $\tilde{f}$ is generally a difficult task. Fortunately, most DP algorithms do not require such an analytical expression. Instead, given any state-action pair, the model is only required to generate a corresponding next state and reward. Constructing this generative model is usually easier.

The expected infinite-horizon discounted return of a state x_0 under a policy h accumulates the rewards obtained by using this policy from x_0:[3]

$$R^h(x_0) = \lim_{K\to\infty} \mathrm{E}_{x_{k+1}\sim\tilde{f}(x_k,h(x_k),\cdot)} \left\{ \sum_{k=0}^{K} \gamma^k \tilde{\rho}(x_k, h(x_k), x_{k+1}) \right\} \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor and the expectation is taken over the stochastic transitions. The notation $x_{k+1} \sim \tilde{f}(x_k, h(x_k), \cdot)$ means that the random variable x_{k+1} is drawn from the density $\tilde{f}(x_k, h(x_k), \cdot)$ at each step k. The goal is to find an optimal policy h^* that maximizes the expected return (1) from every initial state. So, the long-term performance (return) must be maximized using only feedback about the immediate, one-step performance (reward). This is challenging because actions taken in the present potentially affect rewards achieved far into the future, and the immediate reward provides no information about these long-term effects. This is the problem of delayed reward [74]. When the infinite-horizon discounted return is used

[2] As already mentioned, control-theoretic notations are used instead of artificial intelligence notations. For instance, in the artificial intelligence literature on DP/RL , the state is usually denoted by s, the state space by S, the action by a, the action space by A, and the policy by π.

[3] We assume that the MDP and the policies h have suitable properties to ensure that the return and the Bellman equations in the sequel are well-defined. See e.g., [10] and Appendix A of [9] for a discussion of these properties.

and under certain technical assumptions on the elements of the MDP , there exists at least one stationary deterministic optimal policy [10].

The discount factor can intuitively be seen as a way to encode an increasing uncertainty about rewards that will be received in the future. From a mathematical point of view, discounting ensures that, given bounded rewards, the returns will always be bounded. Choosing γ often involves a tradeoff between the quality of the solution and the convergence rate of the DP/RL algorithm. Some important DP/RL algorithms have a rate of convergence proportional to γ, so they converge faster when γ is smaller (this is the case e.g., for model-based value iteration, see Sections 2.2 and 4.1). However, if γ is too small, the solution may be unsatisfactory because it does not sufficiently take into account rewards obtained after a large number of steps.

Instead of discounting the rewards, they can also be averaged over time, or they can simply be added together without weighting [35]. It is also possible to use a finite-horizon return, in which case optimal policies and the optimal value function depend on the time step k. Only infinite-horizon discounted returns, leading to time-invariant optimal policies and value functions, are considered in this chapter.

Policies can be conveniently characterized using their value functions. Two types of value functions exist: state-action value functions (Q-functions) and state value functions (V-functions). The Q-function of a policy h is the return when starting in a given state, applying a given action, and following the policy h thereafter:

$$Q^h(x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma R^h(x') \right\} \tag{2}$$

The optimal Q-function is defined as the best Q-function that can be obtained by any possible policy:[4]

$$Q^*(x,u) = \max_h Q^h(x,u) \tag{3}$$

A policy that selects for every state an action with the largest optimal Q-value:

$$h^*(x) = \arg\max_u Q^*(x,u) \tag{4}$$

is optimal (it maximizes the return). A policy that maximizes a Q-function in this way is said to be *greedy* in that Q-function. Here, as well as in the sequel, if multiple maximizing actions are encountered when computing greedy policies for some state, any of these actions can be chosen. So, finding an optimal policy can be done by first finding Q^*, and then computing a greedy policy in Q^*.

A central result in DP and RL is the *Bellman optimality equation*:

$$Q^*(x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma \max_{u'} Q^*(x',u') \right\} \tag{5}$$

[4] Note that, for the simplicity of notation, we implicitly assume that the maximum exists in (3) and in similar equations in the sequel. When the maximum does not exist, the 'max' operator should be replaced by 'sup', and the theory remains valid. For the computation of greedy actions in (4) and in similar equations in the sequel, the maximum must exist in order to ensure the existence of a greedy policy; this can be guaranteed under certain technical assumptions.

This equation gives a recursive characterization of Q^*: the optimal value of taking action u in state x is the expected sum of the immediate reward and of the discounted optimal value achievable in the next state. The Q-function Q^h of a policy h is also characterized by a Bellman equation, given as follows:

$$Q^h(x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma Q^h(x', h(x')) \right\} \tag{6}$$

which states that the value of taking action u in state x under the policy h is the expected sum of the immediate reward and of the discounted value achieved by h from the next state.

The V-function $V^h : X \to \mathbb{R}$ gives the return when starting from a particular state and following the policy h. It can be computed from the Q-function: $V^h(x) = Q^h(x, h(x))$. The optimal V-function is defined as $V^*(x) = \max_h V^h(x)$ and can be computed from the optimal Q-function: $V^*(x) = \max_u Q^*(x,u)$. The V-functions V^* and V^h satisfy Bellman equations similar to (5) and (6). The optimal policy can be computed from V^*, but the formula to do so is more complicated than (4): it requires a model of the MDP and computing expectations over the stochastic transitions. This hampers the computation of control policies from V-functions, which is a significant drawback in practice. Therefore, in the sequel we will prefer using Q-functions. The disadvantage of Q-functions is that they are more costly to represent, because in addition to x they also depend on u.

In *deterministic* problems, the transition probability function $\tilde{f}$ is replaced by a simpler transition function, $f : X \times U \to X$. This function is obtained from the stochastic dynamics by using a degenerate density $\tilde{f}(x,u,\cdot)$ that assigns all the probability mass to the state $f(x,u)$. The deterministic rewards are completely determined by the current state and action: $\rho(x,u) = \tilde{\rho}(x,u,f(x,u))$. All the formalism given in this section can be specialized to the deterministic case. For instance, the Bellman optimality equation for Q^* becomes:

$$Q^*(x,u) = \rho(x,u) + \gamma \max_{u'} Q^*(f(x,u), u') \tag{7}$$

and the Bellman equation for Q^h becomes:

$$Q^h(x,u) = \rho(x,u) + \gamma Q^h(f(x,u), h(f(x,u))) \tag{8}$$

2.2 Exact value iteration

Value iteration techniques use the Bellman optimality equation to iteratively compute an optimal value function, from which an optimal policy is then derived. As an illustrative example of a DP (model-based) value iteration algorithm, we describe Q-iteration. Let the set of all Q-functions be denoted by $\mathscr{Q}$. Define the Q-iteration mapping $T : \mathscr{Q} \to \mathscr{Q}$, which computes the right-hand side of the Bellman optimality equation (5) for an arbitrary Q-function:

$$[T(Q)](x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma \max_{u'} Q(x',u') \right\} \tag{9}$$

In the deterministic case, the right-hand side of the deterministic Bellman optimality equation (7) should be used instead. It can be shown that T is a contraction with factor $\gamma < 1$ in the infinity norm, i.e., for any pair of functions Q and Q', it is true that $\|T(Q) - T(Q')\|_\infty \le \gamma \|Q - Q'\|_\infty$. The Q-iteration algorithm starts from an arbitrary Q-function Q_0 and in each iteration ℓ updates it using:

$$Q_{\ell+1} = T(Q_\ell) \tag{10}$$

Because T is a contraction, it has a unique fixed point, and from (7), this point is Q^*. This implies that Q-iteration asymptotically converges to Q^* as $\ell \to \infty$. A similar V-iteration algorithm can be given that computes the optimal V-function V^*.

RL (model-free) techniques like Q-learning [87], and Dyna [73] either learn a model, or do not use an explicit model at all. For instance, Q-learning starts from an arbitrary initial Q-function Q_0 and updates it online, using observed transitions $(x_k, u_k, x_{k+1}, r_{k+1})$ [86, 87]. After each transition, the Q-function is updated with:

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k [r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u') - Q_k(x_k,u_k)] \tag{11}$$

where $\alpha_k \in (0,1]$ is the learning rate. The term between square brackets is the temporal difference, i.e., the difference between the current estimate $Q_k(x_k,u_k)$ of the optimal Q-value of (x_k,u_k) and the updated estimate $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u')$. This new estimate is actually a single sample of the expectation on the right-hand side of the Q-iteration mapping (9), applied to Q_k in the state-action pair (x_k,u_k). In this sample, $\tilde{f}(x_k,u_k,x')$ is replaced by the observed next state x_{k+1}, and $\tilde{\rho}(x_k,u_k,x')$ is replaced by the observed reward r_{k+1}. In the discrete-variable case, Q-learning asymptotically converges to Q^* as the number of transitions k approaches infinity, if $\sum_{k=0}^{\infty} \alpha_k^2$ is finite, $\sum_{k=0}^{\infty} \alpha_k$ is infinite, and if all the state-action pairs are (asymptotically) visited infinitely often [29, 87].

The third condition can be satisfied if, among other things, the controller has a non-zero probability of selecting any action in every encountered state; this is called exploration. The controller also has to exploit its current knowledge in order to obtain good performance, e.g., by selecting greedy actions in the current Q-function. A classical way to balance exploration with exploitation is the ε-greedy policy, which selects actions according to:

$$u_k = \begin{cases} \arg\max_u Q(x_k,u) & \text{with probability } 1 - \varepsilon_k \\ \text{a uniformly random action in } U & \text{with probability } \varepsilon_k \end{cases} \tag{12}$$

where $\varepsilon_k \in (0,1)$ is the exploration probability at step k. Usually, ε_k diminishes over time, so that asymptotically, as $Q_k \to Q^*$, the policy used also converges to a greedy, and therefore optimal, policy.

2.3 Exact policy iteration

Policy iteration techniques iteratively evaluate and improve policies [9, 11]. Consider a policy iteration algorithm that uses Q-functions. In every iteration ℓ, such an algorithm computes the Q-function Q^{h_ℓ} of the current policy h_ℓ; this step is called . Then, a new policy $h_{\ell+1}$ that is greedy in Q^{h_ℓ} is computed; this step is called policy improvement. Policy iteration algorithms can be either model-based or model-free; and offline or online.

To implement policy evaluation, define analogously to (9) a *policy evaluation mapping* $T^h : \mathscr{Q} \to \mathscr{Q}$, which applies to any Q-function the right-hand side of the Bellman equation for Q^h. In the stochastic case, the right-hand side of (6) is used, leading to:

$$[T^h(Q)](x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma Q(x', h(x')) \right\} \tag{13}$$

whereas in the deterministic case, the right-hand side of (8) should be used instead. Like the Q-iteration mapping T, T^h is a contraction with a factor $\gamma < 1$ in the infinity norm. A model-based policy evaluation algorithm can be given that works similarly to Q-iteration. This algorithm starts from an arbitrary Q-function Q_0^h and in each iteration τ updates it using:[5]

$$Q_{\tau+1}^h = T^h(Q_\tau^h) \tag{14}$$

Because T^h is a contraction, this algorithm asymptotically converges to Q^h. Other ways to find Q^h include online, sample-based techniques similar to Q-learning, and directly solving the linear system of equations provided by (6) or (8), which is possible when X and U are discrete and the cardinality of $X \times U$ is not very large [9].

Policy iteration starts with an arbitrary policy h_0. In each iteration ℓ, policy evaluation is used to obtain the Q-function Q^{h_ℓ} of the current policy. Then, an improved policy is computed which is greedy in Q^{h_ℓ}:

$$h_{\ell+1}(x) = \arg\max_u Q^{h_\ell}(x,u) \tag{15}$$

The Q-functions computed by policy iteration asymptotically converge to Q^* as $\ell \to \infty$. Simultaneously, the policies converge to h^*.

The main reason for which policy iteration algorithms are attractive is that the Bellman equation for Q^h is linear in the Q-values. This makes policy evaluation easier to solve than the Bellman optimality equation (5), which is highly nonlinear due to the maximization in the right-hand side. Moreover, in practice, offline policy iteration algorithms often converge in a small number of iterations [45, 74], possibly smaller than the number of iterations taken by offline value iteration algorithms. However, this does not necessarily mean that policy iteration is less computationally costly than value iteration. Even though policy evaluation is generally less

[5] A different iteration index τ is used for policy evaluation, because policy evaluation runs in the inner loop of every policy iteration ℓ.

costly than value iteration, *every single* policy iteration requires a complete policy evaluation.

Model-free variants of policy iteration can also be given. SARSA is an online, model-free policy iteration algorithm, proposed in [69] as an alternative to the value-iteration based Q-learning. SARSA starts with an arbitrary initial Q-function Q_0 and updates it using tuples $(x_k, u_k, x_{k+1}, u_{k+1}, r_{k+1})$, as follows:

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k[r_{k+1} + \gamma Q_k(x_{k+1}, u_{k+1}) - Q_k(x_k, u_k)] \tag{16}$$

In contrast to Q-learning (11), which uses the maximal Q-value in the next state to compute the temporal difference, SARSA uses the Q-value of the action actually taken in the next state. This means that SARSA performs online, model-free policy evaluation. To select actions, a greedy policy is combined with exploration, using e.g., the ε-greedy strategy (12). Using a greedy policy means that SARSA implicitly performs a policy improvement at every time step; hence, SARSA is a type of online policy iteration.

Actor-critic algorithms [76] also belong to the class of online policy iteration techniques; they will be presented in Section 5.4. The ‘actor’ is the policy and the ‘critic’ is the value function.

3 The need for approximation in dynamic programming and reinforcement learning

When the state and action spaces of the MDP contain a large or infinite number of elements, value functions and policies cannot be represented exactly. Instead, approximation must be used. Consider, e.g., the algorithms for exact value iteration of Section 2.2. They require to store distinct estimates of the return for every state (in the case of V-functions) or for every state-action pair (Q-functions). When some of the state variables have a very large or infinite number of possible values (e.g., they are continuous), exact storage is no longer possible. Large or continuous action spaces make the representation of Q-functions additionally challenging.

Approximation in DP/RL is not only a problem of representation. Consider e.g., the Q-iteration algorithm of Section 2, which iteratively applies the Q-iteration mapping: $Q_{\ell+1} = T(Q_\ell)$. This mapping would have to be implemented as follows:

$$\textbf{for every } (x,u) \textbf{ do}\text{: } Q_{\ell+1}(x,u) = \mathrm{E}_{x' \sim \tilde{f}(x,u,\cdot)} \left\{ \tilde{\rho}(x,u,x') + \gamma \max_{u' \in U} Q_\ell(x',u') \right\} \tag{17}$$

When the state-action space contains an infinite number of elements, it is impossible to loop over all the state-action pairs in a finite time. Instead, an approximate update has to be used that only considers a finite number of state-action samples. Additionally, the expectation on the right-hand side of (17) cannot be computed exactly, but has to be estimated from a finite number of samples, using Monte Carlo methods. Note that, in many RL algorithms, the Monte Carlo approximation does not appear explicitly, but is performed implicitly while processing samples. Q-learning (11) is such an algorithm.

The maximization over the action variable in (17) must be solved for every sample used in the Monte Carlo estimation. In large or continuous action spaces, this maximization is a potentially difficult non-concave optimization problem, which can only be solved approximately. To simplify this problem, many algorithms discretize the action space in a small number of values, compute the value function for all the discrete actions, and find the maximum among these values using enumeration.

Similar difficulties are encountered in policy iteration algorithms; there, the maximization problem has to be solved at the policy improvement step. Policy search algorithms also need to estimate returns using a finite number of samples, and must find the best policy in the class considered, which is a potentially difficult optimization problem. However, this problem only needs to be solved once, unlike the maximization over actions in value iteration and policy iteration, which must be solved for every sample considered. In this sense, policy search methods are less affected from the maximization difficulties than value iteration or policy iteration.

In deterministic MDP s, the Monte-Carlo estimation is not needed, but sample-based updates and approximate maximization are still required.

4 Approximate value iteration

For value iteration in large or continuous-space MDP s, the value function has to be approximated. Linearly parameterized approximators make it easier to analyze the theoretical properties of the resulting DP/RL algorithms. Nonlinearly parameterized approximators like neural networks have better representation power than linear parameterizations; however, the resulting DP/RL algorithms are more difficult to analyze.

Consider for instance a linearly parameterized approximator for the Q-function. Such an approximator has n basis functions (BF s) $\phi_1, \dots, \phi_n : X \times U \to \mathbb{R}$, and is parameterized by a vector[6] of n parameters $\theta \in \mathbb{R}^n$. Given a parameter vector θ, approximate Q-values are computed with:

$$\widehat{Q}(x,u) = \sum_{l=1}^{n} \phi_l(x,u)\theta_l = \phi^{\mathrm{T}}(x,u)\theta \tag{18}$$

where $\phi(x,u) = [\phi_1(x,u), \dots, \phi_n(x,u)]^{\mathrm{T}}$. The parameter vector θ thus provides a compact (but approximate) representation of a Q-function. Examples of linearly parameterized approximators include crisp discretization [7, 80] (see Example 1), multilinear interpolation [14], Kuhn triangulation [55], and Gaussian radial BF s [51, 80] (see Example 2).

[6] All the vectors used in this chapter are column vectors.

In this section, we describe algorithms for model-based and model-free approximate value iteration. Then, we describe convergence guarantees for approximate value iteration, and apply a representative algorithm to an example.

4.1 Approximate model-based value iteration

This section describes the approximate Q-iteration algorithm with a general parametric approximator, which is an extension of exact Q-iteration algorithm in Section 2.2. Recall that exact Q-iteration starts from an arbitrary Q-function Q_0 and in each iteration ℓ updates the Q-function using $Q_{\ell+1} = T(Q_\ell)$, where T is the Q-iteration mapping (9).

Approximate Q-iteration parameterizes the Q-function using a parameter vector $\theta \in \mathbb{R}^n$. It requires two other mappings in addition to T. The *approximation* mapping $F : \mathbb{R}^n \to \mathscr{Q}$ produces an approximate Q-function $\widehat{Q} = F(\theta)$ for a given parameter vector θ. This Q-function is used as an input to the Q-iteration mapping T. The *projection* mapping $P : \mathscr{Q} \to \mathbb{R}^n$ computes a parameter vector θ such that $F(\theta)$ is as close as possible to a target Q-function Q, e.g., in a least-squares sense. Projection is used to obtain a new parameter vector from the output of the Q-iteration mapping. So, approximate Q-iteration starts with an arbitrary (e.g., identically 0) parameter vector θ_0, and updates this vector in every iteration ℓ using the composition of the mappings P, T, and F:

$$\theta_{\ell+1} = (P \circ T \circ F)(\theta_\ell) \tag{19}$$

Of course, the results of F and T cannot be fully computed and stored. Instead, $P \circ T \circ F$ can be implemented as a single entity, or sampled versions of the F and T mappings can be applied. Once a satisfactory parameter vector θ^* (ideally, a fixed point of the composite mapping $P \circ T \circ F$) has been found, the following policy can be used:

$$\widehat{h}^*(x) = \arg\max_u [F(\theta^*)](x,u) \tag{20}$$

Figure 2 illustrates approximate Q-iteration and the relations between the various mappings, parameter vectors, and Q-functions considered by the algorithm.

We use the notation $[F(\theta)](x,u)$ to refer to the Q-function $F(\theta)$ evaluated at the state-action pair (x,u). For instance, a linearly parameterized approximator (18) would lead to $[F(\theta)](x,u) = \phi^{\mathrm{T}}(x,u)\theta$. The notation $[P(Q)]_l$ refers to the lth component in the parameter vector $P(Q)$.

A similar formalism can be given for approximate V-iteration, which is more popular in the literature [19, 25–27, 55, 80]. Many results from the literature deal with the discretization of continuous-variable problems [19, 25, 27, 55]. Such discretizations are not necessarily crisp, but can use interpolation procedures, which lead to linearly parameterized approximators of the form (18).

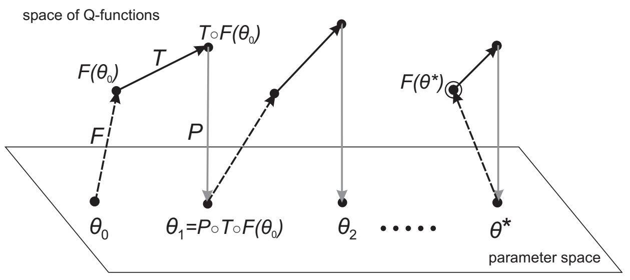


Fig. 2 A conceptual illustration of approximate Q-iteration. In every iteration, F is applied to the current parameter vector to obtain an approximate Q-function, which is then passed through T. The result of T is projected back onto the parameter space with P. Ideally the algorithm converges to a fixed point θ^*, which leads back to itself when passed through $P \circ T \circ F$. The solution of approximate Q-iteration is the Q-function $F(\theta^*)$.

4.2 Approximate model-free value iteration

Approximate model-free value iteration has also been extensively studied [21, 23, 28, 30, 52, 60, 71, 77, 79]. The Q-learning algorithm is the most popular, and has been combined with a variety of approximators, among which:

- linearly parameterized approximators, encountered under several names such as interpolative representations [77] and soft state aggregation [71];
- fuzzy rule-bases [23, 28, 30], which can also be linear in the parameters;
- nonlinearly parameterized approximators such as neural networks [43] and self-organizing maps [79].

The most straightforward way to integrate approximation in Q-learning is by using gradient updates of the parameter vector [74]:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \max_{u'} \widehat{Q}_k(x_{k+1}, u') - \widehat{Q}_k(x_k, u_k) \right] \frac{\partial}{\partial \theta_k} \widehat{Q}_k(x_k, u_k)$$

where the Q-function is parameterized by θ and the term in square brackets is an approximation of the temporal difference (see again (11)). With linearly parameterized approximation (18), this update simplifies to:

$$\theta_{k+1} = \theta_k + \alpha_k \left[r_{k+1} + \gamma \max_{u'} \left(\phi^{\mathrm{T}}(x_{k+1}, u') \theta_k \right) - \phi^{\mathrm{T}}(x_k, u_k) \theta_k \right] \phi(x_k, u_k)$$

Some algorithms for approximate model-free value iteration work offline and require a batch of samples collected in advance. A good example is fitted Q-iteration, which uses ensembles of regression trees (a nonparametric approximator) to represent the Q-function [21]. Fitted Q-iteration belongs to the class of approximate Q-iteration algorithms. It replaces the exact Q-iteration mapping T in (19) by an approximation derived from the available samples, and the projection mapping P by a process that derives a new ensemble of regression trees in every iteration, in order to best approximate the current Q-function. Neural fitted Q-iteration is a similar algorithm, but it approximates the Q-function using neural networks instead of ensembles of regression trees [67].

4.3 Convergence and the role of non-expansive approximators

An important question in approximate DP/RL is whether the approximate solution computed by the algorithm converges, and, if it does converge, how far the convergence point is from the optimal solution. Convergence is important because it makes the algorithm is more amenable to analysis and meaningful performance guarantees.

The convergence proofs for approximate value iteration often rely on contraction mapping arguments. Consider for instance approximate Q-iteration, given by (19). The Q-iteration mapping T is a contraction in the infinity norm with factor $\gamma < 1$, as already mentioned in Section 2.2. If the composite mapping $P \circ T \circ F$ of approximate Q-iteration is also a contraction, i.e., $\|(P \circ T \circ F)(\theta) - (P \circ T \circ F)(\theta')\|_\infty \leq \gamma' \|\theta - \theta'\|_\infty$ for all θ, θ' and for a $\gamma' < 1$, then approximate Q-iteration asymptotically converges to a unique fixed point, which we denote by θ^*.

One way to make $P \circ T \circ F$ a contraction is to ensure that F and P are non-expansions, i.e., that $\|F(\theta) - F(\theta')\|_\infty \leq \|\theta - \theta'\|_\infty$ for all θ, θ' and that $\|P(Q) - P(Q')\|_\infty \leq \|Q - Q'\|_\infty$ for all Q, Q' [26]. In this case, the contraction factor of $P \circ T \circ F$ is the same as that of T: $\gamma' = \gamma < 1$. Under these conditions, as we will describe next, suboptimality bounds can be derived on the solution obtained.

Denote by $\mathscr{F}_{F \circ P} \subset \mathscr{Q}$ the set of fixed points of the composite mapping $F \circ P$ (this set is assumed non-empty). Define $\sigma_{\mathrm{QI}}{}^* = \min_{Q' \in \mathscr{F}_{F \circ P}} \|Q^* - Q'\|_\infty$, the minimum distance between Q^* and any fixed point of $F \circ P$.[7] This distance characterizes the representation power of the approximator; the better the representation power, the closer the nearest fixed point of $F \circ P$ will be to Q^*, and the smaller $\sigma_{\mathrm{QI}}{}^*$ will be. The convergence point θ^* of approximate Q-iteration satisfies the following suboptimality bounds [26, 80]:

$$\|Q^* - F(\theta^*)\|_\infty \leq \frac{2\sigma_{\mathrm{QI}}{}^*}{1-\gamma} \tag{21}$$

$$\|Q^* - Q^{\widehat{h}^*}\|_\infty \leq \frac{4\gamma\sigma_{\mathrm{QI}}{}^*}{(1-\gamma)^2} \tag{22}$$

where $Q^{\widehat{h}^*}$ is the Q-function of a policy $\widehat{h}^*$ that is greedy in $F(\theta^*)$ (20). Equation (21) gives the suboptimality bound of the approximately optimal Q-function, whereas (22) gives the suboptimality bound of the resulting, approximately optimal policy. The latter may be more relevant in practice. The following relationship between the policy suboptimality and the Q-function suboptimality was used to obtain

[7] If the minimum does not exist, then $\sigma_{\mathrm{QI}}{}^*$ should be taken as small as possible so that there still exists a $Q' \in \mathscr{F}_{F \circ P}$ with $\|Q^* - Q'\|_\infty \leq \sigma_{\mathrm{QI}}{}^*$.

(22), and is also valid in general:

$$\|Q^* - Q^h\|_\infty \leq \frac{2\gamma}{(1-\gamma)} \|Q^* - Q\|_\infty \tag{23}$$

where the policy h is greedy in the (arbitrary) Q-function Q.

In order to take advantage of these theoretical guarantees, P and F should be non-expansions. When F is linearly parameterized (18), it is fairly easy to ensure its non-expansiveness by normalizing the BF s ϕ_l, so that for every x and u, we have $\sum_{l=1}^{n} \phi_l(x,u) = 1$. Ensuring that P is non-expansive is more difficult. For instance, the most natural choice for P is a least-squares projection:

$$P(Q) = \arg\min_{\theta} \sum_{l_s=1}^{n_s} |Q(x_{l_s}, u_{l_s}) - [F(\theta)](x_{l_s}, u_{l_s})|^2 \tag{24}$$

for some set of samples $\{(x_{l_s}, u_{l_s}) \mid l_s = 1, \ldots, n_s\}$, where ties in the '$\arg\min$' can be broken arbitrarily. Unfortunately, such a projection can in general be an expansion, and examples of divergence when using it have been given [80, 88]. One way to make P non-expansive is to choose exactly $n_s = n$ samples (for instance, the centers of the BF s), and require that $\phi_{l_s}(x_{l_s}, u_{l_s}) = 1$ and $\phi_{l_s'}(x_{l_s}, u_{l_s}) = 0$ for $l_s \neq l_s'$. Then, the projection mapping (24) simplifies to an assignment that associates each parameter with the Q-value of the corresponding sample:

$$[P(Q)]_{l_s} = Q(x_{l_s}, u_{l_s}) \tag{25}$$

This mapping is clearly non-expansive. More general (but still restrictive) conditions on the BF s under which convergence and near-optimality are guaranteed are given in [80].

In the area of approximate model-free value iteration (which belongs to approximate RL), many approaches are heuristic and do not guarantee convergence [23, 28, 30, 52, 79]. Those that do guarantee convergence use linearly parameterized approximators [21,60,71,77], and often employ conditions related to the non-expansiveness properties above, e.g., for Q-learning [71, 77], or for sample-based batch V-iteration [60].

Another important theoretical property of algorithms for approximate DP and RL is consistency. In model-based value iteration, and more generally in DP , an algorithm is consistent if the approximate value function converges to the optimal one as the approximation accuracy increases [19, 25, 70]. In model-free value iteration, and more generally in RL , consistency is usually understood as the convergence to a well-defined solution as the number of samples increases. The stronger result of convergence to an optimal solution as the approximation accuracy also increases is proven in [60, 77].

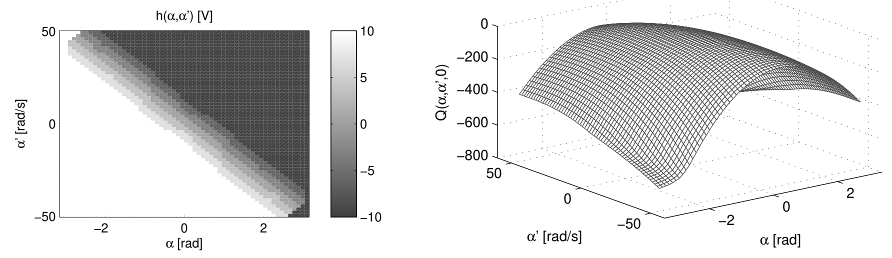
Example 1 (Grid Q-iteration for a DC motor). Consider a second-order discrete-time model of a DC motor:

$$
\begin{aligned}
x_{k+1} &= f(x_k, u_k) = Ax_k + Bu_k \\
A &= \begin{bmatrix} 1 & 0.0049 \\ 0 & 0.9540 \end{bmatrix}, \quad B = \begin{bmatrix} 0.0021 \\ 0.8505 \end{bmatrix}
\end{aligned}
\tag{26}
$$

This model is obtained by discretizing a continuous-time model of the DC motor, which was determined by first-principles modeling of the real DC motor. The discretization is performed with the zero-order-hold method, using a sampling time of $T_\mathrm{s} = 0.005\,\mathrm{s}$. Using saturation, the position $x_{1,k} = \alpha$ is bounded to $[-\pi, \pi]$ rad, the velocity $x_{2,k} = \dot{\alpha}$ to $[-16\pi, 16\pi]$ rad/s. The control input u_k is constrained to $[-10, 10]$ V. A quadratic regulation problem has to be solved, which is described by the following reward function:

$$
\begin{aligned}
r_{k+1} &= \rho(x_k, u_k) = -x_k^\mathrm{T} Q_\mathrm{rew} x_k - R_\mathrm{rew} u_k^2 \\
Q_\mathrm{rew} &= \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix}, \quad R_\mathrm{rew} = 0.01
\end{aligned}
\tag{27}
$$

With this reward function, a good policy will drive the state (close) to 0, while also minimizing the magnitude of the states along the trajectory and the control effort. The discount factor is chosen $\gamma = 0.95$, which is sufficient to produce a good control policy. Figure 3 presents a near-optimal solution to this problem, computed using the convergent and consistent fuzzy Q-iteration algorithm [14] with an accurate approximator.



(a) A near-optimal policy.

(b) Slice through a near-optimal Q-function, for $u = 0$.

Fig. 3 A near-optimal solution for the DC motor.

As an example of approximate value iteration, we develop a Q-iteration algorithm that relies on a gridding of the state space and on a discretization of the action space into a set of finitely many values, $U_\mathrm{d} = \{u_1, \dots, u_M\}$. For this problem, three discrete actions are sufficient to find an acceptable stabilizing policy, $U_\mathrm{d} = \{-10, 0, 10\}$. The state space is gridded (partitioned) into a set of N disjoint rectangles. Let X_i be the surface of the ith rectangle in this partition, with $i = 1, \dots, N$. The Q-function approximator represents distinct slices through the Q-function, one for each of the discrete actions. For a given action, the approximator assigns the same Q-values for all the states in X_i. This corresponds to a linearly parameterized approximator with binary-valued (0 or 1) BF s over the state-discrete action space $X \times U_\mathrm{d}$:

$$\phi_{[i,j]}(x,u) = \begin{cases} 1 & \text{if } x \in X_i \text{ and } u = u_j \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

where $[i, j]$ denotes the single-dimensional index corresponding to i and j, and can be computed with $[i, j] = i + (j-1)N$. Note that because the rectangles are disjoint, exactly one BF is active at any point of $X \times U_\mathrm{d}$.

To derive the projection mapping P, the least-squares projection (24) is used, taking as state-action samples the cross product of the sets $\{x_1, \ldots, x_N\}$ and U_d, where x_i denotes the center of the $i$the rectangle X_i. These samples satisfy the conditions to simplify P to an assignment of the form (25):

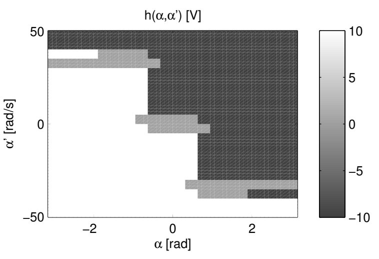
$$[P(Q)]_{[i,j]} = Q(x_i, u_j) \tag{29}$$

Using the linearly parameterized approximator (18) with the BF s (28) and the projection (29) yields the grid Q-iteration algorithm. Because F and P are non-expansions, this algorithm is convergent.

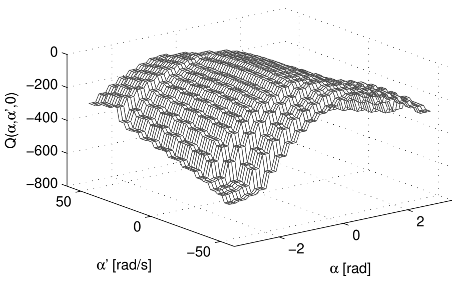
To apply grid Q-iteration to the DC motor problem, two different grids over the state space are used: a coarse grid, with 20 equidistant bins on each axis (leading to $20^2 = 400$ rectangles); and a fine grid, with 100 equidistant bins on each axis (leading to $100^2 = 10000$ rectangles). The algorithm is considered convergent when the maximum amount by which any parameter changes between two consecutive iterations does not exceed $\varepsilon_\mathrm{QI} = 0.001$. For the coarse grid, convergence occurs after 160 iterations, and for the fine grid, after 118. This shows that the number of iterations to convergence is not monotonously increasing with the number of parameters. The finer grid may help convergence because it can achieve a better accuracy. Representative state-dependent slices through the resulting Q-functions (obtained by setting the action argument u to 0), together with the corresponding policies computed with (20), are given in Figure 4. The accuracy in representing the Q-function is worse for the coarse grid, in Figure 4(b), than for the fine grid, in Figure 4(d). The structure of the policy is more clearly visible in Figure 4(c). Axis-oriented artifacts appear for both grid sizes, due to the limitations of the chosen approximator. For instance, the piecewise-constant nature of the approximator is clearly visible in Figure 4(b).

5 Approximate policy iteration

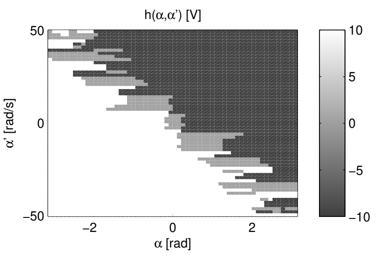
Recall that policy iteration techniques compute in each iteration the value function of the current policy. Then, they compute a new, improved policy, which is greedy in the current value function, and repeat the cycle (see Section 2.3). Approximating the value function is always necessary to apply policy iteration in large and continuous spaces. Sometimes, an explicit representation of the policy can be avoided, by



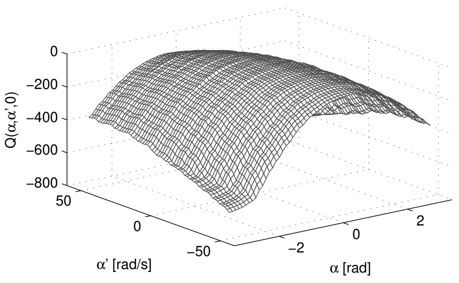
(a) Coarse-grid policy.



(b) Slice through coarse-grid Q-function, for $u = 0$.



(c) Fine-grid policy.



(d) Slice through fine-grid Q-function, for $u = 0$.

Fig. 4 Grid Q-iteration solutions for the DC motor.

computing improved actions on demand from the current value function. Alternatively, the policy can be represented explicitly, in which case policy approximation is generally required. In this case, solving a classical supervised learning problem is necessary to perform policy improvement.

Like in approximate value iteration, the convergence of approximate policy evaluation can be guaranteed more easily with linearly parameterized approximators of the value function. Nonlinearly parameterized approximators, especially neural networks, are also used often in actor-critic algorithms, an important subclass of approximate policy iteration.

Policy evaluation algorithms are discussed first, followed by policy improvement and the resulting policy iteration algorithms. Theoretical guarantees on the solutions obtained are given and a representative algorithm is applied to the DC motor problem of Example 1. Finally, actor-critic techniques are presented.

5.1 Approximate policy evaluation

Some of the most powerful algorithms for approximate policy evaluation combine linearly parameterized approximators of the value function with model-free, least-squares techniques to compute the parameters [9]. We therefore focus on model-free, least-squares policy evaluation in this section. In particular, we discuss the least-squares temporal difference for Q-functions (LSTD-Q) [40] and the least-squares policy evaluation for Q-functions (LSPE-Q) [57]. Model-based, approximate policy evaluation can be derived along the same lines as model-based, approximate value iteration, see Section 4.1.

Assume for now that X and U have a finite number of elements. LSTD-Q and LSPE-Q solve a projected form of the Bellman equation (6): [8]

$$\widehat{Q}^h = P^w(T^h(\widehat{Q}^h)) \tag{30}$$

where P^w performs a weighted least-squares projection onto the space of representable Q-functions, i.e., the space $\left\{\phi^{\mathrm{T}}(x,u)\theta \mid \theta \in \mathbb{R}^n\right\}$ spanned by the BF s. Here, $w : X \times U \to [0,1]$ is the weight function, which is always interpreted as a probability distribution over the state-action space and must therefore satisfy $\sum_{x,u} w(x,u) = 1$. Figure 5 illustrates the projected Bellman equation.

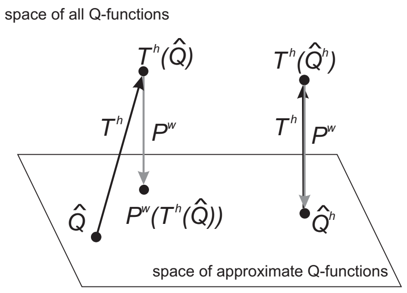


Fig. 5 A conceptual illustration of the projected Bellman equation. Applying T^h and then P^w to an ordinary approximate Q-function leads to a different point in the space of approximate Q-functions (left). In contrast, applying T^h and then P^w to the fixed point $\widehat{Q}^h$ of the projected Bellman equation leads back into the same point (right).

The projection P^w is defined by:

$$[P^w(Q)](x,u) = \phi^{\mathrm{T}}(x,u)\theta^{\ddagger}, \text{ where}$$
$$\theta^{\ddagger} = \arg\min_{\theta} \sum_{(x,u)\in X\times U} w(x,u)\left|\phi^{\mathrm{T}}(x,u)\theta - Q(x,u)\right|^2$$

The function w controls the distribution of the error between a Q-function and its projection, and therefore indirectly controls the accuracy of the solution $\widehat{Q}^h$ of the projected Bellman equation, via (30).

By writing the projected Bellman equation (30) in a matrix form, it can eventually be transformed into a linear equation in the parameter vector (see [9,40] for details):

[8] A multi-step version of this equation can also be given. In this chapter, we only consider the single-step case.

$$\Gamma \theta^h = \gamma \Lambda \theta^h + z \tag{31}$$

where $\Gamma, \Lambda \in \mathbb{R}^{n\times n}$ and $z \in \mathbb{R}^n$. The original, high-dimensional Bellman equation (6) has thus been replaced by the low-dimensional linear system (31). A solution θ^h of this system can be used to find an approximate Q-function with (18).

The matrices Γ, Λ and the vector z can be estimated from transition samples. Consider a set of samples $\{(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}} \sim f(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, \cdot), r_{l_\mathrm{s}} = \rho(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}, x'_{l_\mathrm{s}})) \,|\, l_\mathrm{s} = 1, \ldots, n_\mathrm{s}\}$, constructed by drawing state-action samples (x,u) and then computing corresponding next states and rewards. The probability distribution of the state-action samples is given by the weight function w. The estimates of Γ, Λ, and z are updated with:

$$\begin{aligned}
&\Gamma_0 = 0, \quad \Lambda_0 = 0, \quad z_0 = 0\\
&\Gamma_{l_\mathrm{s}} = \Gamma_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\phi^\mathrm{T}(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\\
&\Lambda_{l_\mathrm{s}} = \Lambda_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}})\phi^\mathrm{T}(x'_{l_\mathrm{s}}, h(x'_{l_\mathrm{s}}))\\
&z_{l_\mathrm{s}} = z_{l_\mathrm{s}-1} + \phi(x_{l_\mathrm{s}}, u_{l_\mathrm{s}}) r_{l_\mathrm{s}}
\end{aligned} \tag{32}$$

LSTD-Q processes the entire batch of samples using (32) and then solves the equation:

$$\frac{1}{n_\mathrm{s}} \Gamma_{n_\mathrm{s}} \widehat{\theta}^h = \gamma \frac{1}{n_\mathrm{s}} \Lambda_{n_\mathrm{s}} \widehat{\theta}^h + \frac{1}{n_\mathrm{s}} z_{n_\mathrm{s}} \tag{33}$$

to find an approximate parameter vector $\widehat{\theta}^h$. Asymptotically, as $n_\mathrm{s} \to \infty$, it is true that $\frac{1}{n_\mathrm{s}}\Gamma_{n_\mathrm{s}} \to \Gamma$, $\frac{1}{n_\mathrm{s}}\Lambda_{n_\mathrm{s}} \to \Lambda$, and $\frac{1}{n_\mathrm{s}} z_{n_\mathrm{s}} \to z$. Therefore, $\widehat{\theta}^h \to \theta^h$ when $n_\mathrm{s} \to \infty$. The parameter vector $\widehat{\theta}^h$ obtained by LSTD-Q gives an approximate Q-function via (18), which can then be used to perform policy improvement. Note that the division by n_s, although not necessary from a formal point of view, helps to increase the numerical stability of the algorithm.

An alternative to LSTD-Q is LSPE-Q , which starts with an arbitrary initial parameter vector θ_0 and updates it with:

$$\begin{aligned}
&\theta_{l_\mathrm{s}} = \theta_{l_\mathrm{s}-1} + \alpha(\theta^{\ddagger}_{l_\mathrm{s}} - \theta_{l_\mathrm{s}-1}), \text{ where:}\\
&\frac{1}{l_\mathrm{s}} \Gamma_{l_\mathrm{s}} \theta^{\ddagger}_{l_\mathrm{s}} = \gamma \frac{1}{l_\mathrm{s}} \Lambda_{l_\mathrm{s}} \theta_{l_\mathrm{s}-1} + \frac{1}{l_\mathrm{s}} z_{l_\mathrm{s}}
\end{aligned} \tag{34}$$

and where α is a step size parameter. To ensure its invertibility, Γ can be initialized to a small multiple of the identity matrix.

The linear systems in (33) and (34) can be solved in several ways, e.g., (i) by matrix inversion, (ii) by Gaussian elimination, or, (iii) by incrementally computing the inverse with the Sherman-Morrison formula. Although for the derivation above it was assumed that X and U are countable, the updates (32), together with LSTD-Q and LSPE-Q , can be applied also in uncountable (e.g., continuous) state-action spaces. Note also that, when the BF vector $\phi(x,u)$ is sparse,[9] the computational efficiency of the updates (32) can be improved by exploiting this sparsity.

In order to guarantee the asymptotic convergence of LSPE-Q to θ^h, the weight (probability of being sampled) of each state-action pair, $w(x,u)$, must be identical to the steady-state probability of this pair along an infinitely-long trajectory generated with the policy h [9]. In contrast, LSTD-Q (33) may have meaningful solutions for many weight functions w, which can make it more robust in practice. An advantage of LSPE-Q over LSTD-Q stems from the incremental nature of LSPE-Q , which means it can benefit from a good initial value of the parameters.

Analogous least-squares algorithms can be given to compute linearly parameterized, approximate V-functions [9]. Recall however that, as explained in Section 2.1, when V-functions are used it is more difficult to compute greedy policies, and therefore to perform policy improvements.

Gradient-based versions of policy evaluation can also be given, using linearly parameterized approximators [72, 81] or nonlinearly parameterized approximators such as neural networks [1, 20]. When combined with linearly parameterized approximators, gradient-based algorithms usually require more samples than least-squares algorithms to achieve a similar accuracy [36, 91].

Note that the only requirement imposed on the approximator by the convergence guarantees of approximate policy evaluation is its linearity in the parameters. In contrast, approximate value iteration imposes additional requirements to ensure that the approximate value iteration mapping is a contraction (Section 4.3).

5.2 Policy improvement. Approximate policy iteration

To obtain a policy iteration algorithm, a method to perform policy improvement is required in addition to approximate policy evaluation. Consider first the case in which the policy is not explicitly represented. Instead, greedy actions are computed on demand from the value function, for every state where a control action is required, using e.g., in the case of Q-functions:

$$h_{\ell+1}(x) = \arg\max_u \widehat{Q}^{h_\ell}(x,u) \tag{35}$$

where ℓ is the iteration index. The policy is then implicitly defined by the value function. If only a small, discrete set of actions is considered, the maximization in the policy improvement step can be solved by enumeration. In this case, policy improvement is exact. For instance, the algorithm obtained by combining exact policy improvement with policy evaluation by LSTD-Q is least-squares policy iteration (LSPI) [39, 40].

[9] The BF vector is sparse, e.g., when the discrete-action approximator described in the upcoming Example 2 is used. This is because the BF vector contains zeros for all the discrete actions that are different from the current discrete action.

Policies can also be approximated, e.g., using a parametric approximator with a parameter vector $\vartheta \in \mathbb{R}^{\mathscr{N}}$. For instance, a linearly parameterized policy approximator uses a set of state-dependent BF s $\varphi_1, \dots, \varphi_{\mathscr{N}} : X \to \mathbb{R}$ and approximates the policy with:[10]

$$\widehat{h}(x) = \sum_{i=1}^{\mathscr{N}} \varphi_i(x)\vartheta_i = \varphi^{\mathrm{T}}(x)\vartheta \tag{36}$$

where $\varphi(x) = [\varphi_1(x), \dots, \varphi_{\mathscr{N}}(x)]^{\mathrm{T}}$. For simplicity, the parameterization (36) is only given for scalar actions, but it can easily be extended to the case of multiple action variables. For this policy parameterization, approximate policy improvement can be performed by solving the linear least-squares problem:

$$\vartheta_{\ell+1} = \arg\min_{\vartheta} \sum_{i_{\mathrm{s}}=1}^{\mathscr{N}_{\mathrm{s}}} \left\| \varphi^{\mathrm{T}}(x_{i_{\mathrm{s}}})\vartheta - \arg\max_{u} \phi^{\mathrm{T}}(x_{i_{\mathrm{s}}}, u)\theta_\ell \right\|_2^2 \tag{37}$$

to find an improved policy parameter vector $\vartheta_{\ell+1}$, where $\{x_1, \dots, x_{\mathscr{N}_{\mathrm{s}}}\}$ is a set of samples for policy improvement. In this formula, $\arg\max_u \phi^{\mathrm{T}}(x_{i_{\mathrm{s}}}, u)\theta_\ell = \arg\max_u \widehat{Q}^{\widehat{h}_\ell}(x_{i_{\mathrm{s}}}, u)$ is an improved, greedy action for the sample $x_{i_{\mathrm{s}}}$; notice that the policy $\widehat{h}_\ell$ is now also an approximation.

In sample-based policy iteration, instead of waiting with policy improvement until a large number of samples have been processed and an accurate approximation of the Q-function for the current policy has been obtained, policy improvements can also be performed after a small number of samples. Such a variant is sometimes called *optimistic* policy iteration [9, 11]. In the extreme, fully optimistic case, a policy that is greedy in the current value function is applied at every step. If the policy is only improved once every several steps, the method is partially optimistic. One instance in which optimistic updates are useful is when approximate policy iteration is applied online, since in that case the policy should be improved often. Optimistic variants of approximate policy iteration can be derived e.g., using gradient-based policy evaluation [20], least-squares policy evaluation [31, 32]. For instance, approximate SARSA is in fact a type of optimistic policy iteration [33] (see also Section 2.3).

Instead of using the Bellman equation to compute the value function of a policy, this value function can also be estimated by Monte Carlo simulations. This is the approach taken in [41], where Q-functions obtained by Monte Carlo policy evaluation are used to obtain improved policies represented as support vector classifiers.

[10] Calligraphic notation is used to differentiate variables related to policy approximation from variables related to value function approximation. So, the policy parameter is ϑ and the policy BF s are denoted by φ, whereas the value function parameter is θ and the value function BF s are denoted by ϕ. Furthermore, the number of policy parameters and BF s is $\mathscr{N}$, and the number of samples for policy approximation is $\mathscr{N}_{\mathrm{s}}$.

5.3 Theoretical guarantees

As long as the policy evaluation and improvement errors are bounded, approximate PI algorithms eventually produce policies with a bounded suboptimality. We formalize these convergence results, which apply to general (possibly nonlinearly parameterized) approximators.

Consider the general case in which both the value functions and the policies are approximated for *non-optimistic* policy iteration. Consider also the case in which Q-functions are used. Assume that the error in every policy evaluation step is bounded by ε_Q:

$$\|\widehat{Q}^{\widehat{h}_\ell} - Q^{\widehat{h}_\ell}\|_\infty \leq \varepsilon_Q, \quad \text{for any } \ell \geq 0$$

and the error in every policy improvement step is bounded by ε_h, in the following sense:

$$\|T^{\widehat{h}_{\ell+1}}(\widehat{Q}^{\widehat{h}_\ell}) - T(\widehat{Q}^{\widehat{h}_\ell})\|_\infty \leq \varepsilon_h, \quad \text{for any } \ell \geq 0$$

where $T^{\widehat{h}_{\ell+1}}$ is the policy evaluation mapping for the improved (approximate) policy, and T is the Q-iteration mapping (9). Then, approximate policy iteration eventually produces policies with performances that lie within a bounded distance from the optimal performance [40]:

$$\limsup_{\ell\to\infty} \left\|\widehat{Q}^{\widehat{h}_\ell} - Q^*\right\|_\infty \leq \frac{\varepsilon_h + 2\gamma\varepsilon_Q}{(1-\gamma)^2} \tag{38}$$

For an algorithm that performs exact policy improvements, such as LSPI , $\varepsilon_h = 0$ and the bound is tightened to:

$$\limsup_{\ell\to\infty} \|\widehat{Q}^{h_\ell} - Q^*\|_\infty \leq \frac{2\gamma\varepsilon_Q}{(1-\gamma)^2} \tag{39}$$

where $\|\widehat{Q}^{h_\ell} - Q^{h_\ell}\|_\infty \leq \varepsilon_Q$, for any $\ell \geq 0$. Note that computing ε_Q (and, when approximate policies are used, computing ε_h) may be difficult in practice, and the existence of these bounds may require additional assumptions on the MDP .

These guarantees do not imply the convergence to a fixed policy. For instance, both the value function and policy parameters might converge to limit cycles, so that every point on the cycle yields a policy that satisfies the bound. Similarly, when exact policy improvements are used, the value function parameter may oscillate, implicitly leading to an oscillating policy. This is a disadvantage with respect to approximate value iteration, which can be guaranteed to converge monotonically to its fixed point Section 4.3.

Similar results hold when V-functions are used instead of Q-functions [11].

Optimistic policy iteration improves the policy before an accurate value function is available. Because the policy evaluation error can be large, the performance guarantees given above are not useful in the optimistic case. The behavior of optimistic policy iteration has not been properly understood yet, and can be very complicated. It can e.g., exhibit a phenomenon called chattering, whereby the value function converges to a stationary function, while the policy sequence oscillates, because the limit of the value function parameter corresponds to multiple policies [9, 11].

Example 2 (LSPI for the DC motor). In this example, LSPI will be applied to the DC motor problem of Example 1. The action space is again discretized into the set $U_\mathrm{d} = \{-10, 0, 10\}$, which contains $M = 3$ actions. Only these discrete actions are allowed into the set of samples. A number of N normalized Gaussian radial basis functions (RBFs) $\bar{\phi}_i : X \to \mathbb{R}$, $i = 1, \ldots, N$, are used to approximate over the state space. The RBF s are defined as follows:

$$\bar{\phi}_i(x) = \frac{\phi_i'(x)}{\sum_{i'=1}^{N} \phi_{i'}'(x)}, \qquad \phi_i'(x) = \exp\left[-\frac{(x_1 - c_{i,1})^2}{b_{i,1}^2} - \frac{(x_2 - c_{i,2})^2}{b_{i,2}^2}\right] \tag{40}$$

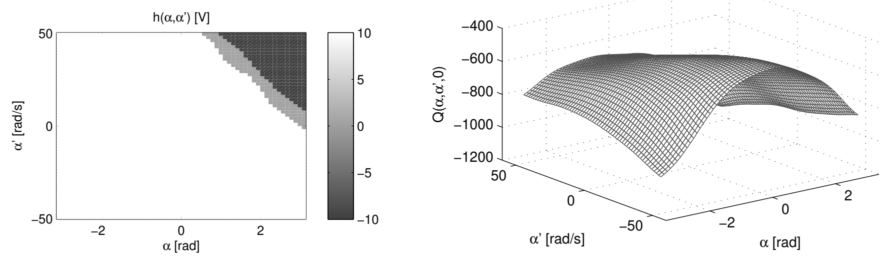
where ϕ_i' are (non-normalized) Gaussian axis-parallel RBF s, $(c_{i,1}, c_{i,2})$ is the center of the ith RBF , and $(b_{i,1}, b_{i,2})$ is its radius. The centers of the RBF s are arranged on an equidistant 9×9 grid in the state space. The radii of the RBF s along each dimension are taken identical to the distance along that dimension between two adjacent RBF s; this yields a smooth interpolation of the Q-function over the state space. The RBF s are replicated for every discrete action, and to compute the state-discrete action BF s, all the RBF s that do not correspond to the current discrete action are taken equal to 0. Approximate Q-values can then be computed with $\widehat{Q}(x, u_j) = \phi^\mathrm{T}(x, u_j)\, \theta$, for the state-action BF vector:

$$\phi(x, u_j) = [\underbrace{0, \ldots, 0}_{u_1}, \ldots, 0, \underbrace{\bar{\phi}_1(x), \ldots, \bar{\phi}_N(x)}_{u_j}, 0, \ldots, \underbrace{0, \ldots, 0}_{u_M}]^\mathrm{T} \in \mathbb{R}^{NM}$$

and a parameter vector $\theta \in \mathbb{R}^n$ with $n = NM = 3N$.

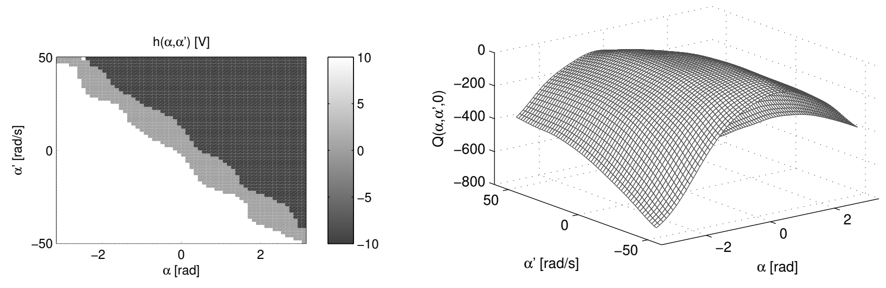
First, LSPI with exact policy improvements is applied, starting from an initial policy h_0 that is identically equal to -10 throughout the state space. The same set of $n_\mathrm{s} = 7500$ samples is used in every LSTD-Q policy evaluation. The samples are random, uniformly distributed over the state-discrete action space $X \times U_\mathrm{d}$. To illustrate the results of LSTD-Q , Figure 6 presents the first improved policy found by LSPI , h_1, and its approximate Q-function, computed with LSTD-Q .

The complete LSPI algorithm converged in 11 iterations. Figure 7 shows the resulting policy and Q-function, which are good approximations of the near-optimal solution in Figure 3. Compared to the results of grid Q-iteration in Figure 4, LSPI needed fewer BF s (9×9 rather than 400 or 10000) and was able to find a better approximation of the policy. This is mainly because the Q-function is largely smooth (see Figure 3(b)), which means it can be represented well using the wide RBF s considered. In contrast, the grid BF s give a discontinuous approximate Q-function, which is less appropriate for this problem. Although certain types of continuous BF s can be used with Q-iteration, using wide RBF s such as these is unfortunately not possible, because they do not satisfy the assumptions for convergence, and indeed lead to divergence when they are too wide. A disadvantage of these wide RBF s is that they fail to properly identify the policy nonlinearities in the top-left and bottom-right corners of Figure 3(a), and the corresponding changes in the Q-function.



(a) Policy. (b) Slice through the Q-function, for $u = 0$.

Fig. 6 An early policy and its approximate Q-function, for LSPI with exact policy improvements.



(a) Policy. (b) Slice through the Q-function, for $u = 0$.

Fig. 7 Results of LSPI with exact policy improvements on the DC motor.

Another observation is that LSPI converges in a significantly fewer iterations than grid Q-iteration did in Example 1 (namely, 11 iterations for LSPI , instead of the 160 iterations taken by grid Q-iteration with the coarse grid, and of the 118 iterations taken with the fine grid). Such a convergence rate advantage of policy iteration over value iteration is often observed in practice. Note however that, while LSPI did converge faster, it was actually more computationally intensive than grid Q-iteration: it required approximately 105 s to run, whereas grid Q-iteration only required 0.5 s for the coarse grid and 6 s for the fine grid.[11] This is mainly because the cost of policy evaluation with LSTD-Q is at least quadratic in the number $n = NM$ of state-action BF s (see the updates (32)). In contrast, the cost of every grid Q-iteration is only $O(n\log(N))$, when binary search is used to locate the position of a state on the grid.

Next, LSPI with *approximate policies* and approximate policy improvements is applied. The policy approximator is (36) and uses the same RBF s as the Q-function approximator ($\varphi_i = \bar{\phi}_i$). As before, $N_s = 7500$ samples are used for policy evaluation. Note that the approximate policy produces continuous actions, which must be quantized (into discrete actions belonging to U_d) before performing policy evalua-

[11] For all the experiments in this chapter, the algorithms are run in MATLAB 7, on a PC with an Intel T2400 CPU and 2GB of RAM.

tion, because the Q-function approximator only works for discrete actions. Approximate policy improvement is performed with (37), using a set of $\mathcal{N}_s = 2500$ random, uniformly distributed state samples. The same set is used in every iteration.

In this experiment, both the Q-functions and the policies *oscillate* in the steady state of the algorithm, with a period of 2 iterations. The execution time until the oscillation was detected was 104 s. The differences between the two distinct policies and Q-functions on the limit cycle are too small to be noticed in a figure. Instead, Figure 8 shows the evolution of the policy parameter that changes the most in steady state. Its oscillation is clearly visible. The appearance of oscillations may be related to the fact that the weaker suboptimality bound (38) applies when approximate policies are used, rather than the stronger bound (39), which applied to the experiment with exact policy improvements.

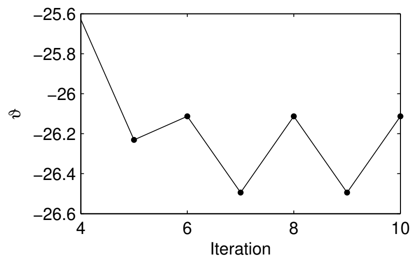
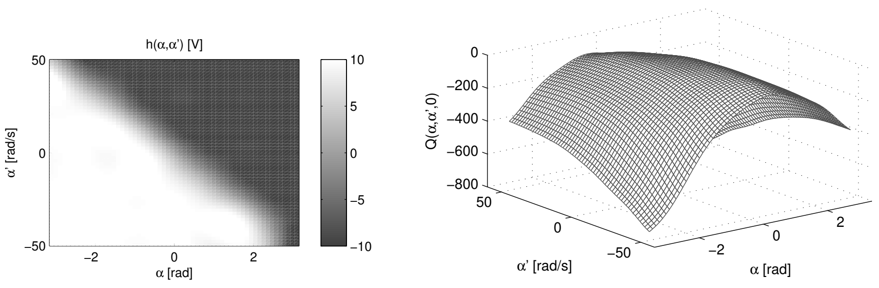


Fig. 8 The variation of one of the policy parameters, starting with the 4th iteration and until the oscillation was detected.

Figure 9 presents one of the two policies from the limit cycle, and one of the Q-functions. The policy and Q-function have a similar accuracy to those computed with exact policy improvements, even though in this experiment the solution oscillated instead of converging to a stationary value. The approximate policy has the added advantage that it produces continuous actions.



(a) Policy. (b) Slice through the Q-function, for $u = 0$.

Fig. 9 Results of LSPI with approximate policy improvement on the DC motor.

5.4 Actor-critic algorithms

Actor-critic algorithms are a special case of online approximate policy iteration. They were introduced in [3] and have been investigated often since then [5, 6, 12, 38, 56, 76]. In typical actor-critic methods, both the policy and the value function are approximated using differentiable approximators (often neural networks [44, 61, 64]), and updated using gradient rules. The critic is the approximate value function (typically a V-function), and the actor is the approximate policy.

Next, a typical actor-critic algorithm is formalized. Denote by $\widehat{V}(x;\theta)$ the approximate V-function, parameterized by $\theta \in \mathbb{R}^N$, and by $\widehat{h}(x;\vartheta)$ the approximate policy, parameterized by $\vartheta \in \mathbb{R}^{\mathscr{N}}$. We use the notation $\widehat{V}(x;\theta)$ (and respectively, $\widehat{h}(x;\vartheta)$) to make explicit the dependence of the parameter vector θ (and respectively, ϑ). Because the algorithm does not distinguish between the value functions of the different policies, the value function notation is not superscripted by the policy. After each transition from x_k to x_{k+1}, the temporal difference $\delta_{\text{TD},k} = r_{k+1} + \gamma\widehat{V}(x_{k+1};\theta_k) - \widehat{V}(x_k;\theta_k)$ is computed. This temporal difference is analogous to the temporal difference for Q-functions, used e.g., in Q-learning (11). It is a sample of the difference between the right-hand and left-hand sides of the Bellman equation for the policy V-function:

$$V(x) = \mathrm{E}_{x' \sim \tilde{f}(x,h(x),\cdot)} \left\{ \rho(x,h(x),x') + V(x') \right\} \tag{41}$$

Since the exact values of the current state, $V(x_k)$, and of the next state, $V(x_{k+1})$ are not available, they are replaced by their approximations.

Once the temporal difference $\delta_{\text{TD},k}$ is available, the policy and V-function parameters are updated with:

$$\theta_{k+1} = \theta_k + \alpha_{\text{C}} \frac{\partial \widehat{V}(x_k;\theta_k)}{\partial \theta} \delta_{\text{TD},k} \tag{42}$$

$$\vartheta_{k+1} = \vartheta_k + \alpha_{\text{A}} \frac{\partial \widehat{h}(x_k;\vartheta_k)}{\partial \vartheta} [u_k - \widehat{h}(x_k;\vartheta_k)]\, \delta_{\text{TD},k} \tag{43}$$

where α_{C} and α_{A} are learning rates (step sizes) for the critic and the actor, respectively, and the notation $\frac{\partial \widehat{V}(x_k;\theta_k)}{\partial \theta}$ means that the derivative $\frac{\partial \widehat{V}(x;\theta)}{\partial \theta}$ is evaluated for the state x_k and the parameter θ_k (and analogously in (43)). In the critic update (42), the temporal difference takes the place of the prediction error $V(x_k) - \widehat{V}(x_k;\theta_k)$, where $V(x_k)$ is the exact value of x_k given the current policy. Since this exact value is not available, it is replaced by the estimate $r_{k+1} + \gamma\widehat{V}(x_{k+1};\theta_k)$ offered by the Bellman equation (41), thus leading to the temporal difference. In the actor update (43), the actual action u_k applied at step k can be different from the action $\widehat{h}(x_k;\vartheta_k)$ indicated by the policy. This change of the action indicated by the policy is the form taken by exploration in the actor-critic algorithm. When the exploratory action u_k leads to a positive temporal difference, the policy is adjusted towards this action. Conversely, when $\delta_{\text{TD},k}$ is negative, the policy is adjusted away from u_k. This is because, like in the critic update, the temporal difference is interpreted as a correction of the predicted performance, so that e.g., if the temporal difference is positive, the obtained performance is considered better than the predicted one.

An important advantage of actor-critic algorithms stems from the fact that their policy updates are incremental and do not require the computation of greedy actions. This means that it is not necessary to solve a difficult optimization problem over the action variable, and continuous actions are easy to handle. The convergence of actor-critic methods is not guaranteed in general. Some actor-critic algorithms employing a value function approximator that is related in a specific way to the policy approximator are provably convergent [6, 37, 38]. Note that, because they use gradient-based updates, all actor-critic algorithms can remain stuck in locally optimal solutions.

6 Finding value function approximators automatically

Parametric approximators of the value function play an important role in approximate value iteration and approximate policy iteration, as seen in Sections 4 and 5. Given the functional form of such an approximator, the DP/RL algorithm computes its parameters. There still remains the problem of finding a good functional form, well suited to the problem at hand. In this section, we consider linearly parameterized approximators such as (18), in which case a good set of BF s has to be found. This focus is motivated by the fact that, in the literature, most methods to find value function approximators are given in this linear setting. Also note that many approaches require a discrete and not too large action space, and focus their effort on finding good state-dependent BF s.

The BF s can be designed in advance, in which case two approaches are possible. The first approach is to design the BF s so that a uniform resolution is obtained over the entire state space (for V-functions) or over the entire state-action space (for Q-functions). Unfortunately, such an approach suffers from the curse of dimensionality: the complexity of a uniform-resolution approximator grows exponentially with the number of state (and possibly action) variables. The second approach is to focus the resolution in certain parts of the state(-action) space, where the value function has a more complex shape, or where it is more important to approximate it accurately. Prior knowledge about the shape of the value function or about the importance of certain areas of the state(-action) space is necessary in this case. Unfortunately, such prior knowledge is often non-intuitive and very difficult to obtain without actually computing the value function.

A more general alternative is to find BF s automatically, rather than designing them. Such an approach should provide BF s suited to each particular problem. BF s can be either constructed offline [47,51], or adapted while the DP/RL algorithm is running [55,65]. Since convergence guarantees typically rely on a fixed set of BF s, adapting the BF s while running the DP/RL algorithm leads to a loss of these guarantees. Convergence guarantees can be recovered by ensuring that BF adaptation is stopped after a finite number of updates; fixed-BF proofs can then be applied to guarantee asymptotic convergence [21].

In the remainder of this section, we give a brief overview of available techniques to find BF s automatically. Resolution refinement techniques are discussed first, followed by BF optimization, and by other techniques for automatic BF discovery.

6.1 Resolution refinement

Resolution refinement techniques start with a few BF s (a coarse resolution) and then refine the BF s as the need arises. These techniques can be further classified in two categories:

- Local refinement (splitting) techniques evaluate whether a particular area of the state space (corresponding to one or several neighboring BF s) has a sufficient accuracy, and add new BF s when the accuracy is deemed insufficient. Such techniques have been proposed e.g., for Q-learning [65, 66, 84], for V-iteration [55], for Q-iteration [53, 82], and for policy evaluation [27].
- Global refinement techniques evaluate the global accuracy of the representation, and refine the BF s if the accuracy is deemed insufficient. All the BF s can be refined uniformly [19], or the algorithm can decide which areas of the state space require more resolution [55]. For instance, in [19, 55], global refinement is applied to V-iteration, while in [77] it is used for Q-learning.

A variety of criteria are used to decide when the BF s should be refined. In [55], an overview of typical criteria is given, together with a comparison between them in the context of V-iteration. For instance, local refinement in a certain area can be performed:

- when the value function is not (approximately) constant in that area [55];
- when the value function is not (approximately) linear in that area [53, 55];
- when the Bellman error (the error between the left-hand and right-hand sides of the Bellman equation, see the upcoming Section 6.2) is large in that area [27];
- or using various other heuristics [65, 82, 84].

Global refinement can be performed e.g., until a desired solution accuracy is met [19]. The approach in [55] works in discrete-action problems, and refines the areas where the V-function is poorly approximated *and* that affect other areas where the actions dictated by the policy change. This approach globally identifies the areas of the state space that must be approximated more accurately in order to find a good policy.

Resolution refinement techniques increase the memory and computational expenses of the DP/RL algorithm whenever they increase the resolution. Care must be taken to prevent the memory and computation expenses from becoming prohibitive. This is an important concern both in approximate DP and in approximate RL . Equally important in approximate RL are the restrictions imposed on resolution refinement by the limited amount of data available. Increasing the power of the approximator means that more data will be required to compute an accurate solution, so the resolution cannot be refined to arbitrary levels for a given amount of data.

6.2 Basis function optimization

Basis function optimization techniques search for the best placement and shape of a (usually fixed) number of BF s. Consider e.g., the linear parameterization (18) of the Q-function. To optimize the n BF s, they can be parameterized by a vector of BF parameters ξ that encodes their locations and shapes. For instance, a radial BF is characterized by its center and width. Denote the parameterized BF s by $\varphi_l(x;\xi) : X \times U \to \mathbb{R}$, $l = 1, \dots, n$, to highlight their dependence on ξ. The BF optimization algorithm searches for an optimal parameter vector ξ^* that optimizes a certain criterion related to the accuracy of the value function representation.

Many optimization techniques can be applied to compute the BF parameters. For instance, gradient-based optimization has been used for temporal difference [71] and least-squares temporal difference algorithms [51]. The cross-entropy method has been applied to least-squares temporal difference [51] and Q-iteration algorithms [15].

The most widely used optimization criterion is the Bellman error, also called Bellman residual [51, 71]. This is a measure of the extent to which the approximate value function violates the Bellman equation, which would be precisely satisfied by the exact value function. For instance, the Bellman error for an estimate $\widehat{Q}$ of the optimal Q-function Q^* is derived from the Bellman optimality equation, namely (5) in the stochastic case and (7) in the deterministic case. So, for a deterministic MDP , this Bellman error is:

$$\int_X \int_U \left| \widehat{Q}(x,u) - \rho(x,u) - \gamma \max_{u'} \widehat{Q}(f(x,u),u') \right|^2 \mathrm{d}u\,\mathrm{d}x \tag{44}$$

In practice, an approximation of the Bellman error is computed using a finite set of samples. The suboptimality $\|\widehat{Q} - Q^*\|_\infty$ of an approximate Q-function $\widehat{Q}$ is bounded by a constant multiple of the infinity norm of its Bellman error $\|\widehat{Q} - T\widehat{Q}\|_\infty$ [11, 89]. Furthermore, the Q-function suboptimality is related to the policy suboptimality by (23), which means that minimizing the Bellman error is useful in principle. Unfortunately, minimizing the *quadratic* Bellman error (44) may lead to a large *infinity norm* of the Bellman error, so it is unclear whether minimizing (44) leads to a near-optimal approximate Q-function and policy.

Another possible criterion for optimizing the BF s is the performance (returns) of the policy obtained by the DP/RL algorithm [15].

6.3 Other methods for basis function construction

It is possible to construct BF s using various other techniques that are different from resolution refinement and optimization. For instance, in [46, 47], a spectral analysis of the MDP transition dynamics is performed to find the BF s. These BF s are then used in LSPI . Because the BF s represent the underlying topology of the state transitions, they provide a good accuracy in representing the value function.

Many nonparametric approximators can be seen as generating a set of BF s automatically. The number, location, and possibly also the shape of the BF s are not established in advance, but are determined by the nonparametric regression algorithm. For instance, in [21], regression trees are used to represent the Q-function in every iteration of the fitted Q-iteration algorithm. The method to build the regression trees implicitly determines a set of BF s that represent well the Q-function at the current iteration. In [32, 90], kernel-based approximators are used in LSPI . Originally, kernel-based approximation uses a BF for each sample, but in [90] a kernel sparsification procedure automatically determines a reduced number of BF s, and in [32] BF s are added online only when they improve the accuracy. In [60], kernel-based approximators are combined with value iteration. Least-squares support vector machines are applied to policy evaluation by least-squares temporal difference in [31], and to Q-learning in [85]. Support vector regression is used with SARSA in [33]. Self-organizing maps are combined with Q-learning in [79].

Example 3 (Finding RBF s for LSPI in the DC motor problem). Consider again the DC motor problem of Example 1, and its solution found with LSPI in Example 2. As already discussed, the LSPI solution of Figure 7(a) does not properly take into account the nonlinearities of the policy seen in the top-left and bottom-right corners of Figure 3(a). This is because the corresponding variations in the Q-function, seen in Figure 3(b), are not represented well by the wide RBF s employed. To improve the resolution in the corners where the Q-function is not well approximated, a resolution refinement technique could be applied.

An alternative is to parameterize the RBF s (40), and optimize their locations and shapes. In this case, the RBF parameter vector, denoted by ξ, would contain the two-dimensional centers and radii of all the RBF s: $\xi = [c_{1,1}, c_{1,2}, b_{1,1}, b_{1,2}, \ldots, c_{N,1}, c_{N,2}, b_{N,1}, b_{N,2}]^{\mathrm{T}}$. Such an approach is proposed in [51], where the Bellman error is minimized using gradient descent and cross-entropy optimization.

7 Approximate policy search

Algorithms for approximate policy search represent the policy approximately, most often using a parametric approximator. An optimal parameter vector is then sought using optimization techniques. In certain special cases (e.g., when the state space is finite and not too large), the parameterization might exactly represent an optimal policy. However, in general, optimal policies can only be represented approximately.

We consider in this section policy search techniques that do not employ value functions. Such techniques are useful when it is undesirable to compute value functions, e.g., because value-function based techniques fail to obtain a good solution.

Denote by $\widehat{h}(x;\vartheta)$ the approximate policy, parameterized by $\vartheta \in \mathbb{R}^{\mathscr{N}}$. Policy search algorithms search for an optimal parameter vector that maximizes the return $R^{\widehat{h}(x;\vartheta)}$ for all $x \in X$. Three additional types of approximation are necessary to implement a general policy search algorithm (see also Section 3):

1. When X is large or continuous, computing the return for every state is not possible. A practical procedure to circumvent this difficulty requires choosing a finite set X_0 of representative initial states. Returns are estimated only for states in X_0, and the optimization criterion (score) is the weighted average return over X_0 [49,54]:
$$s(\vartheta) = \sum_{x_0 \in X_0} w(x_0) R^{\widehat{h}(x;\vartheta)}(x_0) \tag{45}$$
The representative states are weighted by $w : X_0 \to (0,1]$. The set X_0, together with the weight function w, will determine the performance of the resulting policy. For instance, initial states that are deemed more important can be assigned larger weights. Note that maximizing the returns from states in X_0 only results in an approximately optimal policy, because it cannot guarantee that returns from other states in X are maximal.
2. In the computation of the returns, the infinite sum in (1) has to be replaced by a finite sum over K steps. For discounted returns, a value of K that guarantees a maximum absolute error $\varepsilon_{\text{MC}} > 0$ in estimating the returns is [48]:
$$K = \left\lceil \log_\gamma \frac{\varepsilon_{\text{MC}}(1-\gamma)}{\|\tilde{\rho}\|_\infty} \right\rceil \tag{46}$$
where $\lceil \cdot \rceil$ produces the smallest integer larger than or equal to the argument (ceiling).
3. Finally, in stochastic MDP s, Monte Carlo simulations are required to estimate the expected returns. This procedure is consistent, i.e., as the number of simulations approaches infinity, the estimate converges to the correct expectation. Results from Monte Carlo simulation can be applied to bound the approximation error for a finite number of simulations.

If prior knowledge about a (near-)optimal policy is available, an ad-hoc policy parameterization can be designed. For instance, parameterizations that are linear in the state variables can be used, if it is known that a (near-)optimal policy is a linear state feedback. Ad-hoc parameterizations are typically combined with gradient-based optimization [49,54,68].

When prior knowledge about the policy is not available, a richer policy parameterization has to be used. In this case, the optimization criterion is likely to have many local optima, and may also be non-differentiable. This means that gradient-based algorithms are unsuitable, and global, gradient-free optimization algorithms are required. Examples of such techniques include evolutionary optimization (genetic algorithms in particular), tabu search, pattern search, and the cross-entropy method. For instance, evolutionary computation has been applied to policy search in [2, 18, 24], Chapter 3 of [17], and cross-entropy optimization in [16, 48]. Chapter 4 of [17] describes a method to find a policy with the model-reference adaptive search, which is closely related to the cross-entropy method.

Example 4 (Approximate policy search for the DC motor). Consider again the DC motor problem of Example 1. First, we derive a policy parameterization based on prior knowledge, and apply policy search to this parameterization. Then, a policy parameterization that does not rely on prior knowledge is given, and the results obtained with these two parameterizations are compared. To optimize the parameters, we use the global, gradient-free pattern search optimization [42, 78].[12]

Because the system is linear and the reward function is quadratic, the optimal policy would be a linear state feedback if the constraints on the state and action variables were disregarded [9]. Taking now into account the constraints on the action, we assume that a good approximation of an optimal policy is linear in the state variables, up to the constraints on the action:

$$\widehat{h}(x) = \text{sat}\{\vartheta_1 x_1 + \vartheta_2 x_2, -10, 10\} \tag{47}$$

where 'sat' denotes saturation. In fact, an examination of the near-optimal policy in Figure 3(a) reveals that this assumption is largely correct: the only nonlinearities appear in the top-left and bottom-right corners of the figure, and they are probably due to the constraints on the state variables. We use the parameterization (47) and search for an optimal parameter vector $\vartheta^* = [\vartheta_1^*, \vartheta_2^*]^T$.

A set X_0 of representative states must be selected. To obtain a uniform performance across the state space, we select a regular grid of representative states: $X_0 = \{-\pi, -2\pi/3, -\pi/3..., \dots, \pi\} \times \{-16\pi, -12\pi, -8\pi, \dots, 16\pi\}$, weighted uniformly by $w(x_0) = \frac{1}{|X_0|}$. We impose a maximum error $\varepsilon_{MC} = 0.01$ in the estimation of the return. A bound on the reward function (27) can be computed with:

$$\begin{aligned}\|\rho\|_\infty &= \sup_{x,u} \left| -x^T Q_{\text{rew}} x - R_{\text{rew}} u^2 \right| \\ &= \left| -[\pi \; 16\pi] \begin{bmatrix} 5 & 0 \\ 0 & 0.01 \end{bmatrix} \begin{bmatrix} \pi \\ 16\pi \end{bmatrix} - 0.01 \cdot 10^2 \right| \\ &\approx 75.61\end{aligned}$$

To find the trajectory length K required to achieve the precision ε_{MC}, we substitute the values of ε_{MC}, $\|\rho\|_\infty$, and $\gamma = 0.95$ in (46); this yields $K = 233$. Because the problem is deterministic, simulating multiple trajectories from every initial state is not necessary; instead, a single trajectory from every initial state suffices.

Pattern search is applied to optimize the parameters ϑ, starting with a zero initial value of these parameters. The algorithm is considered convergent when the variation of best score decreases below the threshold $\varepsilon_{PS} = 0.01$ (equal to ε_{MC}). The

[12] We use the pattern search algorithm from the *Genetic Algorithm and Direct Search Toolbox* of MATLAB 7.

resulting policy is shown in Figure 10. As expected, it closely resembles the near-optimal policy of Figure 3(a), with the exception of the nonlinearities in the corners.

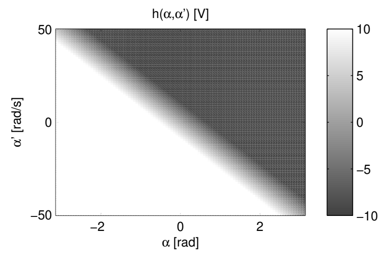


Fig. 10 Results of policy search on the DC motor with the policy parameterization (47). The policy parameter is $\widehat{\vartheta}^* \approx [-16.69, -1]^{\mathrm{T}}$.

The execution time of pattern search was approximately 152 s. This is larger than the execution time of fuzzy Q-iteration in Example 1, which was 6 s for the fine grid and 0.5 s for the coarse grid. It is comparable to the execution time of LSPI in Example 2, which was 105 s when using exact policy improvements, and 104 s with approximate policy improvements. Policy search spends the majority of its execution time estimating the score function (45), which is a computationally expensive operation. For this experiment, the score of 74 different parameter vectors had to be computed until convergence. The complexity can be decreased by taking a smaller number of representative states or larger values for $\varepsilon_{\mathrm{MC}}$ and $\varepsilon_{\mathrm{PS}}$, at the expense of a possible decrease in the control performance.

Consider now the case in which no prior knowledge about the optimal policy is available. In this case, a general policy parameterization must be used. We choose the linear policy parameterization (36), repeated here for easy reference:

$$\widehat{h}(x) = \sum_{i=1}^{\mathscr{N}} \varphi_i(x)\vartheta_i = \varphi^{\mathrm{T}}(x)\vartheta$$

Normalized RBF s (40) are defined, with their centers arranged on an equidistant 7×7 grid in the state space. The radii of the RBF s along each dimension are taken identical to the distance along that dimension between two adjacent RBF s. A number of 49 parameters (for 7×7 RBF s) have to be optimized. This number is larger than for the parameterization (47) derived from prior knowledge, which only had 2 parameters. The same X_0, $\varepsilon_{\mathrm{MC}}$, and $\varepsilon_{\mathrm{PS}} = 0.01$ are used as for the simple parameterization.

The solution obtained by pattern search optimization is shown in Figure 11. Compared to Figure 10, the policy varies more slowly in the linear portion; this is because the wide RBF s used lead to a smooth interpolation. The score obtained by the RBF policy of Figure 11 is -230.69, slightly worse than the score obtained by the policy of Figure 10, which is -229.25.

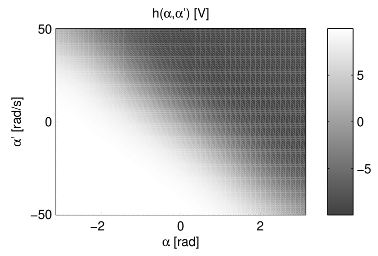


Fig. 11 Results of policy search on the DC motor with the policy parameterization (36).

The algorithm required 30487 s to converge, and had to compute the score of 11440 parameter vectors. As expected, the computational cost is much larger than for the simple parameterization, because many more parameters have to be optimized. This illustrates the benefits of using a policy parameterization that is appropriate for the problem considered. Unfortunately, deriving an appropriate parameterization requires prior knowledge, which is not always available.

8 Comparison of approximate value iteration, policy iteration, and policy search

While a definitive comparison between approximate value iteration, approximate policy iteration, and approximate policy search will depend on the particular algorithms considered, some general remarks can be made.

Algorithms for approximate policy iteration often converge in a smaller number of iterations than algorithms for approximate value iteration, as illustrated in Examples 1 and 2. However, approximate policy evaluation is a difficult problem in itself, which must be solved at each single policy iteration. Since the cost of a policy evaluation may be comparable to the cost of value iteration, it is unclear how the entire policy iteration algorithm compares to value iteration from the point of view of computational cost.

The convergence guarantees for approximate policy evaluation impose less restrictive requirements on the approximator than the guarantees of approximate value iteration; this is an advantage for approximate policy iteration. Namely, for policy evaluation it suffices if the approximator is linearly parameterized (Section 5.3), whereas for value iteration additional properties are required to ensure that the approximate value iteration mapping is a contraction (Section 4.3). Moreover, efficient least-squares algorithms such as LSTD-Q can be used to compute a 'one-shot' solution to the policy evaluation problem. These advantages stem from the *linearity* of the Bellman equation for the value function of a given policy, e.g., (8); whereas the Bellman optimality equation, which characterizes the optimal value function, e.g., (7), is *highly nonlinear* due to the maximization in the right-hand side. Note however that for value iteration, monotonous convergence to a unique solution is usually guaranteed, whereas policy iteration is generally only guaranteed to converge to a sequence of policies that all provide at least a guaranteed level of performance (see Section 5.3).

Approximate policy search is useful in two cases. The first case is when the form of a (near-)optimal policy is known, and only a few parameters need to be determined. In this case, optimization can be used to find a good parameter vector with moderate computational costs. The second case is when, even though prior knowledge is not available, it is undesirable to compute value functions, e.g., because value-function based techniques fail to obtain a good solution or require too restrictive assumptions. In this case, a general policy parameterization can be defined, and a policy search technique that does not rely on value functions can be used to optimize the parameters. Such techniques are usually free from numerical problems – such as divergence to infinity – even when used with general nonlinear parameterizations, which is not the case for value and policy iteration. However, because of its generality, this approach typically incurs large computational costs.

9 Summary and outlook

This chapter has described DP and RL for large or continuous-space, infinite-horizon problems. After introducing the necessary background in exact DP and RL , the need for approximation in DP and RL has been explained, and approximate versions for the three main categories of DP/RL algorithms have been discussed: value iteration, policy iteration, and policy search. Theoretical guarantees have been given and practical algorithms have been illustrated using numerical examples. Additionally, techniques to automatically determine value function approximators have been reviewed, and the three categories of algorithms have been compared.

Approximate DP/RL is a young, but active and rapidly expanding field of research. Many issues in this field remain open. Some of these issues are specific to approximate DP/RL , while others also apply to exact DP and RL .

Next, we discuss some open issues that are specific to approximate DP and RL .

- Automatically finding good approximators is essential in high-dimensional problems, because approximators that provide a uniform accuracy would require too much memory and computation. Adaptive value-function approximators are being extensively studied (Section 6). In policy search, finding approximators automatically is a comparatively under-explored, but promising idea. Nonparametric approximation is an elegant and powerful framework to derive a good approximator from the data [21, 32, 90].
- Continuous-action MDP s are less often studied than discrete-action MDP s, among others because value iteration and policy improvement are significantly more difficult when continuous actions are considered (Section 3). However, for some problems continuous actions are important. For instance, stabilizing a system around an unstable equilibrium requires continuous actions to avoid chattering of the control action, which would otherwise damage the system in the

long run. Continuous actions are easier to handle in actor-critic and policy search algorithms [38, 54, 62].

- Owing to their sample efficiency and relaxed convergence requirements, least-squares techniques for policy evaluation are extremely promising in approximate policy iteration. However, they typically work offline and assume that a large number of samples is used for every policy evaluation. From a learning perspective, it would be interesting to study these techniques in the online case, where the policy must be improved once every few samples, before each policy evaluation has converged. Such optimistic, least-squares policy iteration algorithms are rarely studied in the literature.

Finally, we present several important open issues that apply to both the exact and approximate cases.

- In practice, it is essential to provide guaranteed performance during online RL . Online RL algorithms should ideally guarantee a monotonous increase in their expected performance. Unfortunately, this is generally impossible, because all the online RL algorithms need to explore, i.e., try out actions that may be suboptimal, in order to make sure their performance does not remain stuck in a local optimum. Therefore, weaker requirements could be used, where an overall trend of increased performance is guaranteed, while still allowing for bounded and temporary decreases in performance due to exploration.
- Designing a good reward function is an important and nontrivial step of applying DP and RL . Classical texts on RL recommend to keep the reward function as simple as possible; it should only reward the achievement of the final goal [74]. Unfortunately, a simple reward function often makes online learning very slow, and more information may need to be included in the reward function. Such informative rewards are sometimes called shaping rewards [58]. Moreover, additional, higher-level requirements often have to be considered in addition to the final goal. For instance, in automatic control the controlled state trajectories often have to satisfy requirements on overshoot and the rate of convergence to an equilibrium, etc. Translating such requirements into the ‘language’ of rewards can be very challenging.
- It is important to address problems in which the state signal cannot be measured directly, because such problems often arise in practice. These problems are called partially observable in the DP/RL literature. Algorithms for this type of problem are being extensively researched [4, 34, 50, 59, 63].
- RL has become one of the dominating paradigms for learning in distributed, multi-agent systems [13]. New challenges arise in multi-agent RL , as opposed to the single-agent case. Two major new challenges are that the curse of dimensionality is made worse by the multiple agents present in the system, and that the control actions of the agents must be coordinated in order to reach their intended result. Approximation is an essential, though largely unexplored open issue also in multi-agent RL . This means that a good understanding of single-agent approximate RL is required to develop effective multi-agent RL algorithms.

References

1. Baddeley, B.: Reinforcement learning in continuous time and space: Interference and not ill conditioning is the main problem when using distributed function approximators. IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics **38**(4), 950–956 (2008)
2. Barash, D.: A genetic search in policy space for solving Markov decision processes. In: AAAI Spring Symposium on Search Techniques for Problem Solving under Uncertainty and Incomplete Information. Palo Alto, US (1999)
3. Barto, A.G., Sutton, R.S., Anderson, C.W.: Neuronlike adaptive elements than can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics **13**(5), 833–846 (1983)
4. Baxter, J., Bartlett, P.L.: Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research **15**, 319–350 (2001)
5. Berenji, H.R., Khedkar, P.: Learning and tuning fuzzy logic controllers through reinforcements. IEEE Transactions on Neural Networks **3**(5), 724–740 (1992)
6. Berenji, H.R., Vengerov, D.: A convergent actor-critic-based FRL algorithm with application to power management of wireless transmitters. IEEE Transactions on Fuzzy Systems **11**(4), 478–485 (2003)
7. Bertsekas, D.P.: Adaptive aggregation methods for infinite horizon dynamic programming. IEEE Transactions on Automatic Control **34**(6), 589–598 (1989)
8. Bertsekas, D.P.: Dynamic programming and suboptimal control: A survey from ADP to MPC. European Journal of Control **11**(4–5) (2005). Special issue for the CDC-ECC-05 in Seville, Spain.
9. Bertsekas, D.P.: Dynamic Programming and Optimal Control, vol. 2, 3rd edn. Athena Scientific (2007)
10. Bertsekas, D.P., Shreve, S.E.: Stochastic Optimal Control: The Discrete Time Case. Academic Press (1978)
11. Bertsekas, D.P., Tsitsiklis, J.N.: Neuro-Dynamic Programming. Athena Scientific (1996)
12. Borkar, V.: An actor-critic algorithm for constrained Markov decision processes. Systems & Control Letters **54**, 207–213 (2005)
13. Buşoniu, L., Babuška, R., De Schutter, B.: A comprehensive survey of multi-agent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews **38**(2), 156–172 (2008)
14. Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R.: Consistency of fuzzy model-based reinforcement learning. In: Proceedings 2008 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-08), pp. 518–524. Hong Kong (2008)
15. Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R.: Fuzzy partition optimization for approximate fuzzy Q-iteration. In: Proceedings 17th IFAC World Congress (IFAC-08), pp. 5629–5634. Seoul, Korea (2008)
16. Buşoniu, L., Ernst, D., De Schutter, B., Babuška, R.: Policy search with cross-entropy optimization of basis functions. In: Proceedings 2009 IEEE International Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL-09), pp. 153–160. Nashville, US (2009)
17. Chang, H.S., Fu, M.C., Hu, J., Marcus, S.I.: Simulation-Based Algorithms for Markov Decision Processes. Springer (2007)
18. Chin, H.H., Jafari, A.A.: Genetic algorithm methods for solving the best stationary policy of finite Markov decision processes. In: Proceedings 30th Southeastern Symposium on System Theory, pp. 538–543. Morgantown, US (1998)
19. Chow, C.S., Tsitsiklis, J.N.: An optimal one-way multigrid algorithm for discrete-time stochastic control. IEEE Transactions on Automatic Control **36**(8), 898–914 (1991)
20. Coulom, R.: Feedforward neural networks in reinforcement learning applied to high-dimensional motor control. In: Proceedings 13th International Conference on Algorithmic Learning Theory (ALT-02), pp. 403–414. Lübeck, Germany (2002)

21. Ernst, D., Geurts, P., Wehenkel, L.: Tree-based batch mode reinforcement learning. Journal of Machine Learning Research **6**, 503–556 (2005)
22. Ernst, D., Glavic, M., Capitanescu, F., Wehenkel, L.: Reinforcement learning versus model predictive control: a comparison on a power system problem. IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics **39**(2), 517–529 (2009)
23. Glorennec, P.Y.: Reinforcement learning: An overview. In: Proceedings European Symposium on Intelligent Techniques (ESIT-00), pp. 17–35. Aachen, Germany (2000)
24. Gomez, F.J., Schmidhuber, J., Miikkulainen, R.: Efficient non-linear control through neuroevolution. In: Proceedings 17th European Conference on Machine Learning (ECML-06), *Lecture Notes in Computer Science*, vol. 4212, pp. 654–662. Berlin, Germany (2006)
25. Gonzalez, R.L., Rofman, E.: On deterministic control problems: An approximation procedure for the optimal cost I. The stationary problem. SIAM Journal on Control and Optimization **23**(2), 242–266 (1985)
26. Gordon, G.: Stable function approximation in dynamic programming. In: Proceedings 12th International Conference on Machine Learning (ICML-95), pp. 261–268. Tahoe City, US (1995)
27. Grüne, L.: Error estimation and adaptive discretization for the discrete stochastic Hamilton-Jacobi-Bellman equation. Numerical Mathematics **99**, 85–112 (2004)
28. Horiuchi, T., Fujino, A., Katai, O., Sawaragi, T.: Fuzzy interpolation-based Q-learning with continuous states and actions. In: Proceedings 5th IEEE International Conference on Fuzzy Systems (FUZZ-IEEE-96), pp. 594–600. New Orleans, US (1996)
29. Jaakkola, T., Jordan, M.I., Singh, S.P.: On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation **6**(6), 1185–1201 (1994)
30. Jouffe, L.: Fuzzy inference system learning by reinforcement methods. IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews **28**(3), 338–355 (1998)
31. Jung, T., Polani, D.: Least squares SVM for least squares TD learning. In: Proceedings of 17th European Conference on Artificial Intelligence (ECAI-06), pp. 499–503. Riva del Garda, Italy (2006)
32. Jung, T., Polani, D.: Kernelizing LSPE(λ). In: Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning, pp. 338–345. Honolulu, US (2007)
33. Jung, T., Uthmann, T.: Experiments in value function approximation with sparse support vector regression. In: Proceedings 15th European Conference on Machine Learning (ECML-04), pp. 180–191. Pisa, Italy (2004)
34. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artificial Intelligence **101**, 99–134 (1998)
35. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. Journal of Artificial Intelligence Research **4**, 237–285 (1996)
36. Konda, V.: Actor-critic algorithms. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, US (2002)
37. Konda, V.R., Tsitsiklis, J.N.: Actor-critic algorithms. In: S.A. Solla, T.K. Leen, K.R. Müller (eds.) Advances in Neural Information Processing Systems 12, pp. 1008–1014. MIT Press (2000)
38. Konda, V.R., Tsitsiklis, J.N.: On actor-critic algorithms. SIAM Journal on Control and Optimization **42**(4), 1143–1166 (2003)
39. Lagoudakis, M., Parr, R., Littman, M.: Least-squares methods in reinforcement learning for control. In: Methods and Applications of Artificial Intelligence, *Lecture Notes in Artificial Intelligence*, vol. 2308, pp. 249–260. Springer (2002)
40. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. Journal of Machine Learning Research **4**, 1107–1149 (2003)
41. Lagoudakis, M.G., Parr, R.: Reinforcement learning as classification: Leveraging modern classifiers. In: Proceedings 20th International Conference on Machine Learning (ICML-03), pp. 424–431. Washington, US (2003)
42. Lewis, R.M., Torczon, V.: Pattern search algorithms for bound constrained minimization. SIAM Journal on Optimization **9**(4), 1082–1099 (1999)

43. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning **8**(3/4), 293–321 (1992). Special Issue on Reinforcement Learning.
44. Liu, D., Javaherian, H., Kovalenko, O., Huang, T.: Adaptive critic learning techniques for engine torque and air-fuel ratio control. IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics **38**(4), 988–993 (2008)
45. Madani, O.: On policy iteration as a newton s method and polynomial policy iteration algorithms. In: Proceedings 18th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence AAAI/IAAI-02, pp. 273–278. Edmonton, Canada (2002)
46. Mahadevan, S.: Samuel meets Amarel: Automating value function approximation using global state space analysis. In: Proceedings 20th National Conference on Artificial Intelligence and the 17th Innovative Applications of Artificial Intelligence Conference (AAAI-05), pp. 1000–1005. Pittsburgh, US (2005)
47. Mahadevan, S., Maggioni, M.: Proto-value functions: A Laplacian framework for learning representation and control in Markov decision processes. Journal of Machine Learning Research **8**, 2169–2231 (2007)
48. Mannor, S., Rubinstein, R.Y., Gat, Y.: The cross-entropy method for fast policy search. In: Proceedings 20th International Conference on Machine Learning (ICML-03), pp. 512–519. Washington, US (2003)
49. Marbach, P., Tsitsiklis, J.N.: Approximate gradient methods in policy-space optimization of Markov reward processes. Discrete Event Dynamic Systems: Theory and Applications **13**, 111–148 (2003)
50. McCallum, A.: Overcoming incomplete perception with utile distinction memory. In: Proceedings 10th International Conference on Machine Learning (ICML-93), pp. 190–196. Amherst, US (1993)
51. Menache, I., Mannor, S., Shimkin, N.: Basis function adaptation in temporal difference reinforcement learning. Annals of Operations Research **134**, 215–238 (2005)
52. Millán, J.d.R., Posenato, D., Dedieu, E.: Continuous-action Q-learning. Machine Learning **49**(2-3), 247–265 (2002)
53. Munos, R.: Finite-element methods with local triangulation refinement for continuous reinforcement learning problems. In: Proceedings 9th European Conference on Machine Learning (ECML-97), pp. 170–182. Prague, Czech Republic (1997)
54. Munos, R.: Policy gradient in continuous time. Journal of Machine Learning Research **7**, 771–791 (2006)
55. Munos, R., Moore, A.: Variable-resolution discretization in optimal control. Machine Learning **49**(2-3), 291–323 (2002)
56. Nakamura, Y., Moria, T., Satoc, M., Ishiia, S.: Reinforcement learning for a biped robot based on a CPG-actor-critic method. Neural Networks **20**, 723–735 (2007)
57. Nedić, A., Bertsekas, D.P.: Least-squares policy evaluation algorithms with linear function approximation. Discrete Event Dynamic Systems **13**, 79–110 (2003)
58. Ng, A.Y., Harada, D., Russell, S.: Policy invariance under reward transformations: Theory and application to reward shaping. In: Proceedings 16th International Conference on Machine Learning (ICML-99), pp. 278–287. Bled, Slovenia (1999)
59. Ng, A.Y., Jordan, M.I.: PEGASUS: A policy search method for large MDPs and POMDPs. In: Proceedings 16th Conference in Uncertainty in Artificial Intelligence (UAI-00), pp. 406–415. Palo Alto, US (2000)
60. Ormoneit, D., Sen, S.: Kernel-based reinforcement learning. Machine Learning **49**(2–3), 161–178 (2002)
61. Pérez-Uribe, A.: Using a time-delay actor-critic neural architecture with dopamine-like reinforcement signal for learning in autonomous robots. In: S. Wermter, J. Austin, D.J. Willshaw (eds.) Emergent Neural Computational Architectures Based on Neuroscience, *Lecture Notes in Computer Science*, vol. 2036, pp. 522–533. Springer (2001)
62. Peters, J., Schaal, S.: Natural actor-critic. Neurocomputing **71**, 1180–1190 (2008)
63. Porta, J.M., Vlassis, N., Spaan, M.T., Poupart, P.: Point-based value iteration for continuous POMDPs. Journal of Machine Learning Research **7**, 2329–2367 (2006)

64. Prokhorov, D., Wunsch D.C., I.: Adaptive critic designs. IEEE Transactions on Neural Networks **8**(5), 997–1007 (1997)
65. Ratitch, B., Precup, D.: Sparse distributed memories for on-line value-based reinforcement learning. In: Proceedings 15th European Conference on Machine Learning (ECML-04), *Lecture Notes in Computer Science*, vol. 3201, pp. 347–358. Pisa, Italy (2004)
66. Reynolds, S.I.: Adaptive resolution model-free reinforcement learning: Decision boundary partitioning. In: Proceedings 17th International Conference on Machine Learning (ICML-00), pp. 783–790. Stanford University, US (2000)
67. Riedmiller, M.: Neural fitted Q-iteration – first experiences with a data efficient neural reinforcement learning method. In: Proceedings 16th European Conference on Machine Learning (ECML-05), *Lecture Notes in Computer Science*, vol. 3720, pp. 317–328. Porto, Portugal (2005)
68. Riedmiller, M., Peters, J., Schaal, S.: Evaluation of policy gradient methods and variants on the cart-pole benchmark. In: Proceedings 2007 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-07), pp. 254–261. Honolulu, US (2007)
69. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Tech. Rep. CUED/F-INFENG/TR166, Engineering Department, Cambridge University, UK (1994)
70. Santos, M.S., Vigo-Aguiar, J.: Analysis of a numerical dynamic programming algorithm applied to economic models. Econometrica **66**(2), 409–426 (1998)
71. Singh, S.P., Jaakkola, T., Jordan, M.I.: Reinforcement learning with soft state aggregation. In: G. Tesauro, D.S. Touretzky, T.K. Leen (eds.) Advances in Neural Information Processing Systems 7, pp. 361–368 (1995)
72. Sutton, R.S.: Learning to predict by the method of temporal differences. Machine Learning **3**, 9–44 (1988)
73. Sutton, R.S.: Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In: Proceedings 7th International Conference on Machine Learning (ICML-90), pp. 216–224. Austin, US (1990)
74. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
75. Sutton, R.S., Barto, A.G., Williams, R.J.: Reinforcement learning is adaptive optimal control. IEEE Control Systems Magazine **12**(2), 19–22 (1992)
76. Sutton, R.S., McAllester, D.A., Singh, S.P., Mansour, Y.: Policy gradient methods for reinforcement learning with function approximation. In: S.A. Solla, T.K. Leen, K.R. Müller (eds.) Advances in Neural Information Processing Systems 12, pp. 1057–1063. MIT Press (2000)
77. Szepesvári, C., Smart, W.D.: Interpolation-based Q-learning. In: Proceedings 21st International Conference on Machine Learning (ICML-04), pp. 791–798. Bannf, Canada (2004)
78. Torczon, V.: On the convergence of pattern search algorithms. SIAM Journal on Optimization **7**(1), 1–25 (1997)
79. Touzet, C.F.: Neural reinforcement learning for behaviour synthesis. Robotics and Autonomous Systems **22**(3–4), 251–81 (1997)
80. Tsitsiklis, J.N., Van Roy, B.: Feature-based methods for large scale dynamic programming. Machine Learning **22**(1–3), 59–94 (1996)
81. Tsitsiklis, J.N., Van Roy, B.: An analysis of temporal difference learning with function approximation. IEEE Transactions on Automatic Control **42**(5), 674–690 (1997)
82. Uther, W.T.B., Veloso, M.M.: Tree based discretization for continuous state space reinforcement learning. In: Proceedings 15h National Conference on Artificial Intelligence and 10th Innovative Applications of Artificial Intelligence Conference (AAAI-98/IAAI-98), pp. 769–774. Madison, WI, US (1998)
83. Vrabie, D., Pastravanu, O., Abu-Khalaf, M., Lewis, F.: Adaptive optimal control for continuous-time linear systems based on policy iteration. Automatica **45**(2), 477–484 (2009)
84. Waldock, A., Carse, B.: Fuzzy Q-learning with an adaptive representation. In: Proceedings 2008 IEEE World Congress on Computational Intelligence (WCCI-08), pp. 720–725. Hong Kong (2008)
85. Wang, X., Tian, X., Cheng, Y.: Value approximation with least squares support vector machine in reinforcement learning system. Journal of Computational and Theoretical Nanoscience **4**(7/8), 1290–1294 (2007)

86. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, King's College, Oxford (1989)
87. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning **8**, 279–292 (1992)
88. Wiering, M.: Convergence and divergence in standard and averaging reinforcement learning. In: Proceedings 15th European Conference on Machine Learning (ECML'04), pp. 477–488. Pisa, Italy (2004)
89. Williams, R.J., Baird, L.C.: Tight performance bounds on greedy policies based on imperfect value functions. In: Proceedings 8th Yale Workshop on Adaptive and Learning Systems, pp. 108–113. New Haven, US (1994)
90. Xu, X., Hu, D., Lu, X.: Kernel-based least-squares policy iteration for reinforcement learning. IEEE Transactions on Neural Networks **18**(4), 973–992 (2007)
91. Yu, H., Bertsekas, D.P.: Convergence results for some temporal difference methods based on least-squares. Tech. Rep. LIDS 2697, Massachusetts Institute of Technology, Cambridge, US (2006)