Annual Reviews in Control

Reinforcement Learning for Control: Performance, Stability, and Deep Approximators

Lucian Buşoniu[a], Tim de Bruin[b], Domagoj Tolić[c], Jens Kober[b], Ivana Palunko[d]

[a]*Technical University of Cluj-Napoca, Romania (lucian@busoniu.net)*
[b]*Delft University of Technology, the Netherlands ({t.d.debruin,j.kober}@tudelft.nl)*
[c]*RIT Croatia, Don Frana Bulića 6, 20000 Dubrovnik, Croatia (domagoj.tolic@croatia.rit.edu)*
[d]*University of Dubrovnik, Ćira Carića 4, 20000 Dubrovnik, Croatia (ivana.palunko@unidu.hr)*

Abstract

Reinforcement learning (RL) offers powerful algorithms to search for optimal controllers of systems with nonlinear, possibly stochastic dynamics that are unknown or highly uncertain. This review mainly covers artificial-intelligence approaches to RL, from the viewpoint of the control engineer. We explain how approximate representations of the solution make RL feasible for problems with continuous states and control actions. Stability is a central concern in control, and we argue that while the control-theoretic RL subfield called adaptive dynamic programming is dedicated to it, stability of RL largely remains an open question. We also cover in detail the case where deep neural networks are used for approximation, leading to the field of deep RL, which has shown great success in recent years. With the control practitioner in mind, we outline opportunities and pitfalls of deep RL; and we close the survey with an outlook that – among other things – points out some avenues for bridging the gap between control and artificial-intelligence RL techniques.

Keywords: reinforcement learning, optimal control, deep learning, stability, function approximation, adaptive dynamic programming

1. Introduction

Reinforcement learning (RL) is a model-free framework for solving optimal control problems stated as Markov decision processes (MDPs) (Puterman, 1994). MDPs work in discrete time: at each time step, the controller receives feedback from the system in the form of a state signal, and takes an action in response. Hence, the decision rule is a state feedback control law, called policy in RL. The action changes the system state, possibly in a stochastic manner, and the latest transition is evaluated via a reward function (negative cost). The optimal control objective is then to maximize from each initial state the (expected) cumulative reward, known as value. The problem is thus one of sequential decision-making, so as to optimize the long-term performance. A first advantage of MDP solution techniques is their generality: they can handle nonlinear and stochastic dynamics and nonquadratic reward functions. While MDPs and their solutions classically work for discrete-valued states and actions, this limitation is sidestepped by leveraging numerical function approximation techniques, and such approximate RL algorithms are a main focus of current RL research. Beyond its generality, another crucial advantage of RL is that it is *model-free*: it does not require a model of the system dynamics, or indeed, even the expression of the reward function. Instead, it learns from samples of transitions and rewards, either offline, on a batch of samples obtained in advance from the system, or online, by obtaining the samples directly from the system in closed-loop, simultaneously with learning an optimal controller. Thus, RL is an extremely valuable tool to find (near-)optimal controllers for nonlinear stochastic systems, in cases when the dynamics are either unknown or affected by significant uncertainty.

RL is a large field, and researchers from many backgrounds contribute to it: artificial intelligence (AI), control, robotics, operations research, economics, neuroscience, etc. Many books and survey papers have therefore been published on the topic, from equally varied perspectives: AI, where the classical textbook is that of Sutton and Barto (1998) with the second edition (Sutton and Barto, 2018), but also (Kaelbling et al., 1996; Gosavi, 2009; Szepesvári, 2010; Buşoniu et al., 2010; Wiering and van Otterlo, 2012); control theory (Lewis and Vrabie, 2009; Lewis and Liu, 2012); operations-research flavored optimal control (Bertsekas and Tsitsiklis, 1996; Bertsekas, 2012, 2017; Powell, 2012); robotics (Deisenroth et al., 2011; Kober et al., 2013) etc. Some surveys focus on specific subareas, like policy gradient techniques (Deisenroth et al., 2011; Grondman et al., 2012), function approximation (Geist and Pietquin, 2013), Bayesian formulations of RL (Ghavamzadeh et al., 2015), hierarchical RL (Barto and Mahadevan, 2003), multiagent approaches (Buşoniu et al., 2008), deep RL (Arulkumaran et al., 2017; Li, 2017) and so on.

Against this extensive backdrop, our survey provides several contributions that we explain in the sequel, along with our organization of reviewed methods, shown in Figure 1. Among all the perspectives on RL we focus on two, shown as the two largest rectangles in the figure: AI, since it arguably provides the widest, most general array of algorithms, and control-theoretic, due to our specific interest in this area. Throughout the paper, we use the name approximate, or adaptive, dy-

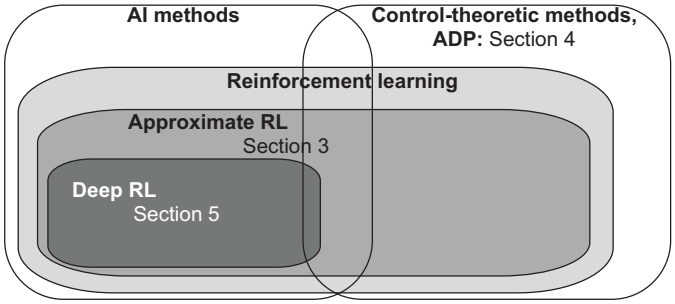

Figure 1: Taxonomy of the approaches reviewed, shown as a Venn diagram.

namic programming (ADP) for control-theoretic methods. We begin in Section 2 by defining the problem and covering basics that are required for all the methods discussed; this section is therefore not shown in Figure 1. In both AI and control, some methods are model-based, white in Figure 1; while RL is model-free, in light gray. We discuss model-based methods only to the extent necessary to understand RL. Note that AI researchers use the name "model-based RL" for algorithms that learn a model from data (effectively performing system identification) and then derive their solution partly using this model. Here, to avoid confusion we call these methods model-learning (while still classifying them as RL), and we reserve the model-based term only for approaches that employ a model from the start.

When RL is applied to e.g. physical systems, the state and action variables are continuous, and due to the generality of the dynamics and reward functions considered, it is usually impossible to derive exact, closed-form representations of the value function or control policy. Function approximation techniques must instead be applied to represent them, and we focus Section 3 on the topic of approximate RL (medium gray). We discuss batch, offline techiques as well as online learning methods of two types. The first type, called temporal-difference learning, together with the batch methods, can be seen as a sample-based implementation of dynamic programming. Half of these methods work in a rather particular way that is atypical in control, called policy iteration, where the long-term values of a fixed policy are found and only then the policy is improved. The second type of online learning comprises policy gradient methods, in which policy parameters are directly optimized from observed data, similarly to extremum seeking in control (Ariyur and Krstic, 2003) but more general.

While these approximate RL methods originate in AI, we explain them for (and from the viewpoint of) the control engineer. With the exception of some works on the relation between RL and model-predictive control (MPC) (Bertsekas, 2005; Ernst et al., 2009; Beuchat et al., 2016; Görges, 2017), such mixed perspectives are usually not taken, and instead overviews focus separately either on AI methods (Kaelbling et al., 1996; Sutton and Barto, 1998; Gosavi, 2009; Szepesvári, 2010; Buşoniu et al., 2010; Wiering and van Otterlo, 2012; Sutton and Barto, 2018) or on ADP approaches (Lewis and Vrabie, 2009; Lewis and Liu, 2012). In contrast, our goal is to take a control-theoretic perspective of AI techniques, and suggest some ways in which the connection between the two fields might be strengthened. With this in mind, we fully dedicate Section 4 to control-theoretic considerations and ADP methods. It turns out that there are fundamental philosophical differences between AI and control, stemming from the fact that AI researchers focus almost exclusively on performance with respect to a freely chosen reward function, while control objectives revolve around stability. While ADP approaches do address stability, many challenges remain, so that in our opinion stable RL is effectively an open area, ripe for novel research.

When RL is combined with a particular type of function approximator, deep neural networks (DNNs), the resulting methods are known as deep reinforcement learning (DRL). This field has recently attracted significant attention and concerted research efforts, due to some impressive results ranging from super-human performance in Atari games (Mnih et al., 2015) and Go (Silver et al., 2017), to motion control in e.g. robotics. Therefore, we dedicate Section 5 to DRL (dark grey in Figure 1). The successes of DRL, combined with the black box nature of DNNs, have led to some practitioners using DRL algorithms without considering the alternatives. We therefore discuss DRL by starting from the basic assumptions underlying DNNs and the optimization procedures used to train them. We then show both the potential and the possible pitfalls of those assumptions in the context of RL, and examine how popular DRL algorithms handle the pitfalls and exploit the potential. This relatively narrow focus on the effects of the assumptions underpinning DRL aims to allow (especially control) practitioners to better evaluate if DRL is the right tool for their task. For a broader view of DRL we refer the reader to the recent reviews (Arulkumaran et al., 2017; Li, 2017).

Section 6 (not shown in the figure) closes the paper by an outlook that gives references to related areas of research, points out some ways of generalizing algorithms for e.g. output feedback, and signals some important open issues in the field.

List of main symbols and notation.

x, u, X, U, r	state, action, state and action spaces, reward
$f, \rho, \bar{f}, \bar{\rho}$	dynamics, rewards, and their deterministic variants
k, n	discrete time step, length of multi-step return
γ, V, Q	discount factor, value function, Q-function
$\pi, \tilde{\pi}$	policy, stochastic policy
V^*, Q^*, π^*	optimal value function, Q-function, policy
$\pi, T^\pi[\cdot]$	policy, Bellman mapping under policy π
$T[\cdot], \mathcal{T}[\cdot]$	Bellman optimality mapping for Q- and V-functions
$\mathbb{Q}, \mathbb{R}, \mathbb{P}$	quadratic state and action penalty, Riccati solution
$\alpha, \ell, t, \varepsilon$	learning rate, major and minor iteration, precision
δ, e, λ	temporal difference, eligibility trace and parameter
$\hat{Q}, \hat{\pi}$	approximate Q-function and policy
ϕ	basis functions
θ, w	Q-function and policy parameters
θ^-, w^-	the same, but for the target networks in deep RL
p, m	number of Q-function and policy parameters
$\mathcal{S}, S, q$	dataset, number of samples, Q-value target
$\boldsymbol{g}$	matrix representation of generic mapping g

d, J	weight or probability, objective function
$\cdot^\top, \nabla\cdot$	transpose, gradient w.r.t. parameters
$\sim, \mathrm{P}(\cdot), E\{\cdot\}$	sampling from distribution, probability, expectation
$\lvert\cdot\rvert, \lVert\cdot\rVert, \lVert\cdot\rVert_\infty$	set cardinality, generic norm, infinity norm

2. Basics of reinforcement learning

These basic concepts and algorithms of RL can be found in standard textbooks (Sutton and Barto, 2018; Bertsekas, 2012; Powell, 2012; Szepesvári, 2010; Buşoniu et al., 2010), so we will only include citations for some notable results and we refer the reader to the textbooks for other details.

2.1. Optimal control problem and its solution

RL solves a discrete-time optimal control problem that is typically formalized as a *Markov decision process* (MDP) (Puterman, 1994). Due to the origins of the field in artificial intelligence, this formalism has some particularities, both in terminology and in some technical choices like using maximization and discounting. Whenever it is useful, we provide insight on AI versus control-theoretic concepts and terminology.

An MDP consists of the state space X of the system, the action (input) space U, the transition function (dynamics) f of the system, and the reward function ρ (negative costs). We leave X and U generic here, although in control they are often real vector spaces. Transitions are usually stochastic, so that, as a result of the action u_k applied in state x_k at discrete time step k, the state changes randomly to x_{k+1}, drawn from $f(x_k, u_k, \cdot)$, which must define a valid probability density. The transition function is therefore a collection of such densities, $f : X \times U \times X \to [0, \infty)$.

As a result of the transition to x_{k+1}, a scalar reward $r_{k+1} = \rho(x_k, u_k, x_{k+1})$ is also received, according to the reward function $\rho : X \times U \times X \to \mathbb{R}$. The reward evaluates the immediate effect of action u_k, but in general does not say anything about its long-term effects. We use notation inspired from control theory; the usual AI notation would be (with varying capitalization) s for state, a for action, T or P for the dynamics, and R for the reward function.

The controller (often called agent in RL) chooses actions according to its policy π, which we take for now to be a deterministic state feedback, $\pi : X \to U$, so that $u_k = \pi(x_k)$. Given a trajectory $x_0, u_0, x_1, u_1, \ldots$ with the associated rewards $r_1, r_2, \ldots$ the infinite-horizon discounted return along this trajectory is:

$$\sum_{k=0}^{\infty} \gamma^k r_{k+1} \tag{1}$$

where the discount factor $\gamma \in (0, 1]$. The value of a policy π from initial state x_0 is the expectation of the return under the stochastic transitions obtained while following π:

$$V^\pi(x_0) = E_{x_{k+1} \sim f(x_k, \pi(x_k), \cdot)} \left\{ \sum_{k=0}^{\infty} \gamma^k \rho(x_k, \pi(x_k), x_{k+1}) \right\} \tag{2}$$

or stated differently, the sum of the returns of all possible trajectories starting from x_0, where each return is weighted by the probability of its trajectory. Functions $V(x)$ are called (state) value functions, or V-functions.

The control objective is to find an optimal policy π^* that attains the maximal value function:[1]

$$V^*(x_0) := \max_\pi V^\pi(x_0), \forall x_0 \tag{3}$$

Note that any policy that attains the maxima in this equation is optimal. An important property of MDPs is that, under appropriate conditions, there exists a deterministic optimal policy that maximizes the value, despite the fact that transitions are stochastic (Bertsekas and Shreve, 1978). Thus, the objective is to maximize the infinite-horizon expected discounted return. Finite-horizon versions are possible, see (Bertsekas, 2017) for details, but we do not cover them here. In control, one would typically solve undiscounted problems ($\gamma = 1$) with unbounded rewards, in which case (stability) conditions must be imposed to ensure that values are bounded and the problem is well posed. In RL, the discount factor is usually taken subunitary and rewards are assumed to be bounded, which makes the value functions well-behaved irrespective of stability concerns. How to analytically reconcile the performance-oriented philosophy of AI-based RL with the stability focus of control is still an open question, and we return to it in Section 4.

Instead of directly using value functions, RL often uses Q-functions $Q : X \times U \to \mathbb{R}$, which fix the initial action:

$$Q^\pi(x, u) = E_{x' \sim f(x,u,\cdot)} \{\rho(x, u, x') + \gamma V^\pi(x')\} \tag{4}$$

Note that we use the prime notation to generically indicate quantities at the next discrete time step, without reference to any particular step k. The intuitive meaning of the Q-value $Q^\pi(x, u)$ is that of expected return when starting from state x, applying the first action u, and following π thereafter. The optimal Q-function Q^* is defined using V^* on the right hand side of (4). The reason for preferring Q-functions is simple: once Q^* is available, an optimal policy can be computed easily, by selecting at each state an action with the largest optimal Q-value:

$$\pi^*(x) \in \arg\max_u Q^*(x, u) \tag{5}$$

When there are multiple maximizing actions, any of them is optimal. In contrast, the formula to compute π^* from V^* is more complicated and – crucially – involves a model, which is usually unavailable in RL. In general, for any Q-function, a policy π that satisfies $\pi(x) \in \arg\max_u Q(x, u)$ is said to be greedy in Q. So, finding an optimal policy can be done by first finding Q^*, and then a greedy policy in this optimal Q-function. Note that the state value functions can be easily expressed in terms of Q-functions, $V^\pi(x) = Q^\pi(x, \pi(x))$, and $V^*(x) = \max_u Q^*(x, u) = Q^*(x, \pi^*(x))$.

The Q-functions Q^π and Q^* are recursively characterized by the *Bellman equations*, which are a consequence of the Q-function definitions, and have central importance for RL algorithms. The Bellman equation for Q^π states that the value of

[1] Whenever the set in which an object ranges is obvious from the context, we leave it implicit. Here, for example, the set is that of all possible policies – functions from x to u.

taking action u in state x under the policy π equals the expected sum of the immediate reward and the discounted value achieved by π in the next state:

$$Q^\pi(x,u) = E_{x'\sim f(x,u,\cdot)}\left\{\rho(x,u,x') + \gamma Q^\pi(x',\pi(x'))\right\} \quad (6)$$

The Bellman optimality equation characterizes Q^*, and states that the optimal Q-value of action u taken in state x equals the sum of the immediate reward and the discounted optimal value obtained by the best action in the next state:

$$Q^*(x,u) = E_{x'\sim f(x,u,\cdot)}\left\{\rho(x,u,x') + \gamma \max_{u'} Q^*(x',u')\right\} \quad (7)$$

We will also interpret the right-hand side of each Bellman equation as a mapping applied to the Q-function, denoted T^π for (6) and simply T for (7). Thus, the Bellman equations may be written as fixed-point relations in the space of Q-functions, $Q^\pi = T^\pi[Q^\pi]$ and $Q^* = T[Q^*]$, but we may also choose to apply these mappings to other, arbitrary Q-functions.

It is instructive to examine the deterministic special case of the framework, as control-theoretic formulations of optimal control are usually deterministic. In that case, the density $f(x,u,\cdot)$ assigns all the probability mass to a unique next state x', leading to deterministic dynamics $x' = \bar{f}(x,u)$, where now $\bar{f} : X \times U \to X$. Since x' is now fixed in the reward function as well, we may also simplify the reward function to $\bar{\rho}(x,u) = \rho(x,u,\bar{f}(x,u))$, with $\bar{\rho} : X \times U \to \mathbb{R}$. The expectation in the V- and Q-function definitions (2) and (4) disappears as a single, deterministic trajectory remains possible, with a return of the form (1). The Bellman optimality equation (7) becomes:

$$Q^*(x,u) = \bar{\rho}(x,u) + \gamma \max_{u'} Q^*(\bar{f}(x,u),u') \quad (8)$$

As an example, consider the linear quadratic case, in which $X = \mathbb{R}^p$, $U = \mathbb{R}^q$, $\bar{f}(x,u) = Ax + Bu$, and $\bar{\rho}(x,u) = -x^\top \mathbb{Q} x - u^\top \mathbb{R} u$ (note again that rewards are equal to negative costs). The Bellman optimality equation boils down to the familiar Riccati equation, which in the discounted case is (Bertsekas, 2012):

$$\mathbb{P} = A^\top[\gamma\mathbb{P} - \gamma^2\mathbb{P}B(\gamma B^\top \mathbb{P} B + \mathbb{R})^{-1}B^\top\mathbb{P}]A + \mathbb{Q}. \quad (9)$$

The optimal policy is a linear state feedback, $\pi^*(x) = Lx$, where

$$L = -\gamma(\gamma B^\top \mathbb{P} B + \mathbb{R})^{-1}B^\top \mathbb{P} A. \quad (10)$$

To avoid introducing unfamiliar notation for symbols with long-established meanings both in RL and control theory, we use different fonts, e.g. Q for Q-function versus $\mathbb{Q}$ for state weights, where the latter font indicates the control-theoretic meaning.

2.2. Offline model-based methods for finite state-action spaces

RL tackles general nonlinear dynamics and nonquadratic rewards, so analytical solutions like in the linear quadratic case are no longer possible, and numerical algorithms must be used. The basic methods readily work for state and action spaces X, U consisting of a finite number of discrete elements. The rationale is that exact representations of the Q-functions and policies are possible in this case. Note that since X is finite and hence countable, the transition function now collects probability mass functions: $f(x,u,x')$ is the probability of transitioning to x' as a result of u in x, and $f : X \times U \times X \to [0,1]$.

We start from two methods that sit at the basis of model-free, RL algorithms: value and policy iteration. These methods work offline and are model-based, i.e. they require the knowledge of f and ρ. For Q-functions, value iteration turns the Bellman optimality equation (7) into an iterative assignment, where the unknown optimal Q-function on the right-hand side is replaced by the current iterate. This procedure is called Q-iteration[2] and shown in Algorithm 1, where the expectation has been written as a sum owing to the countable states, and $\|Q\|_\infty := \max_{x,u}|Q(x,u)|$. If rewards are bounded and the discount factor is subunitary, the updates of this algorithm are contractive and will asymptotically converge to the unique fixed point given corresponding to the optimal Q-function Q^*. In practice, the algorithm is stopped once the iterates no longer change significantly, so it returns an estimate $\hat{Q}^*$ and a corresponding greedy policy. Note that Q-iteration is, in fact, classical dynamic programming, but applied "forward in iterations" rather than backward in time. After a sufficient number of iterations, the steady-state infinite-horizon solution is (approximately) reached.

Algorithm 1 Q-iteration.

Input: f, ρ, γ, threshold ε

1: initialize Q-function, e.g. $Q_0(x,u) = 0\ \forall x,u$
2: **repeat** at every iteration $\ell = 0,1,2,\ldots$
3: **for** every (x,u) pair **do**
4: $Q_{\ell+1}(x,u) = \sum_{x'} f(x,u,x')\left[\rho(x,u,x') + \gamma \max_{u'} Q_\ell(x',u')\right]$
5: **end for**
6: **until** $\|Q_{\ell+1} - Q_\ell\|_\infty \le \varepsilon$

Output: $\hat{Q}^* = Q_{\ell+1}, \hat{\pi}^*(x)$ greedy in $\hat{Q}^*$

Unlike Q-iteration, policy iteration works explicitly on policies, where at each iteration the Q-function of the current policy π_ℓ is computed (policy evaluation), and then a new policy $\pi_{\ell+1}$ is found that is greedy in Q^{π_ℓ} (policy improvement). The procedure is given in Algorithm 2. Assuming policy evaluation is exact, each policy improvement is guaranteed to find a strictly better policy unless it is already optimal, and since in finite spaces there is a finite number of possible policies, the algorithm converges in a finite number of iterations.

Algorithm 2 Policy iteration.

Input: $\gamma, f, \rho,$

1: initialize policy π_0
2: **repeat** at every iteration $\ell = 0,1,2,\ldots$
3: find Q^{π_ℓ}, the Q-function of π_ℓ ▷ policy evaluation
4: $\pi_{\ell+1}(x) = \arg\max_u Q^{\pi_\ell}(x,u), \forall x$ ▷ policy improvement
5: **until** $\pi_{\ell+1} = \pi_{\ell+1}$

Output: $\pi^* = \pi_\ell$, $Q^* = Q^{\pi_\ell}$

[2] The names of important methods and algorithms are underlined.

Algorithm 3 provides an iterative policy evaluation method that, similarly to Q-iteration, turns the Bellman equation (6) into an assignment. This algorithm has similar convergence properties to Q-iteration.

Algorithm 3 Iterative policy evaluation.

Input: π, f, ρ, γ, threshold ε

1: initialize Q-function, e.g. $Q_0(x,u) = 0\ \forall x,u$
2: **repeat** at every iteration $t = 0, 1, 2, \ldots$
3: **for** every (x,u) pair **do**
4: $Q_{t+1}(x,u) = \sum_{x'} f(x,u,x') \left[\rho(x,u,x') + \gamma Q_t(x', \pi(x'))\right]$
5: **end for**
6: **until** $\|Q_{t+1} - Q_t\|_\infty \leq \varepsilon$

Output: $\hat{Q}^\pi = Q_{t+1}$

When there are not too many states and actions, it is also possible to explicitly solve the Bellman equations (6) (for policy evaluation) or (7) (for Q^*), since they are in fact a system of equations with the Q-values as the unknowns. For policy evaluation, the equations are linear so they can be solved relatively cheaply. For the optimal Q-function, they are nonlinear, but with certain relaxations they can still be feasibly solved – see Section 4.2 later on.

2.3. Online temporal-difference RL for finite spaces

We discuss next temporal-difference algorithms, the most popular type of RL. There are many such algorithms, and we select just a few common ones. All methods below are model-free (i.e. they do not need to know f, ρ) and online (i.e. they can work alongside the system). We remain in the finite-space case in this section.

Perhaps the most popular RL algorithm is Q-learning (Watkins and Dayan, 1992). It starts from an arbitrary initial Q-function Q_0, where the index now signifies discrete time k, and updates it using observed state transitions and rewards, i.e. data tuples of the form $(x_k, u_k, x_{k+1}, r_{k+1})$:

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

where $\alpha_k \in (0,1]$ is the learning rate. This update applies an incremental correction to $Q_k(x_k,u_k)$, equal to α_k times the temporal difference in square brackets. The latter difference is between the updated estimate $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u')$ of the optimal Q-value of (x_k,u_k), based on information at the next step, and the current estimate $Q_k(x_k,u_k)$. The updated estimate is a sample of the right-hand side of the Q-iteration update in line 4 of Algorithm 1, in which the expectation over next states is replaced by the observed, random transition sample x_{k+1} and its corresponding received reward r_{k+1}. The temporal difference is therefore (a sample of) the error between the two sides of the Bellman optimality equation (7), and Q-learning can be understood as an online, incremental, stochastic-approximation version of Q-iteration.

As the number of transitions k grows to infinity, Q-learning asymptotically converges to Q^* if (i) $\sum_{k=0}^{\infty} \alpha_k^2$ is finite and $\sum_{k=0}^{\infty} \alpha_k$ is infinite, while (ii) all the state-action pairs are visited infinitely often (Watkins and Dayan, 1992; Jaakkola et al., 1994). Condition (i) is standard in stochastic approximation: it requires that learning rates shrink but not too quickly. A valid example is $\alpha_k = 1/k$. In practice, the learning rate schedule requires tuning, because it influences the convergence rate (number of samples required to perform well).

To satisfy condition (ii), firstly any state must be reachable from any other state (or otherwise, multiple experiments must be run with richly selected initial states). Secondly, it is not sufficient to apply a deterministic policy to obtain the samples, as that would not attempt all the actions in a given state, and may not dynamically reach the entire state space either. Instead, the policy must apply *exploration*: e.g. actions are selected randomly, and each action has nonzero probability at any state. Thus the policy becomes stochastic, $\tilde{\pi} : X \times U \to [0,1]$. A classical choice (Sutton and Barto, 2018) is the ε-greedy policy, which applies a greedy action with probability $1 - \varepsilon$, and otherwise selects an action uniformly randomly, leading to the policy probabilities:

$$\tilde{\pi}(x,u) = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{|U|} & \text{for some } u \in \arg\max_{u'} Q(x,u') \\ \frac{\varepsilon}{|U|} & \text{for the other actions} \end{cases} \quad (12)$$

Here $\varepsilon \in (0,1]$ is the exploration probability, and $|\cdot|$ denotes set cardinality. Another option is Boltzmann, or softmax, exploration, which selects an action u with probability:

$$\tilde{\pi}(x,u) = \frac{e^{Q(x,u)/\tau}}{\sum_{u'} e^{Q(x,u')/\tau}} \quad (13)$$

Here τ is the exploration temperature, and by changing τ the policy can be tuned from fully greedy (in the limit as $\tau \to 0$) to fully random (as $\tau \to \infty$). Importantly, both (12) and (13) strive to balance exploration with the requirement of performing well while controlling the system – exploitation of current knowledge as represented by the Q-function. This exploration-exploitation dilemma is central to all online RL algorithms. Solving it for the two policies above boils down to selecting good schedules ε_k or τ_k, which is a nontrivial tuning problem in practice. Note that exploration in RL is a field in its own right, and we cannot do it justice in this survey; some pointers to major classes of exploration techniques are provided in Section 6.

From a control point of view, exploration is a persistence of excitation condition: even if Q-learning never uses an explicit model, a model is still implicitly present in the Q-values, and persistent excitation is needed to "identify" it. Note also that while in control problems the reward function would be known, Q-learning does not make this assumption and learns about the rewards at the same time as about the dynamics. A final remark is that, as long as conditions (i) and (ii) above are satisfied, Q-learning works no matter what policy is actually applied to control the system. Q-learning can therefore be seen as always evaluating the greedy policy, while the behavior policy can be anything exploratory – a property called being off-policy. Algorithm 4 presents Q-learning with an arbitrary behavior policy.

Algorithm 4 Q-learning.

initialize Q-function, e.g. $Q_0(x, u) = 0\ \forall x, u$
measure initial state x_0
for every time step $k = 0, 1, 2, \dots$ **do**
    select action u_k with an exploratory policy
    apply u_k, measure next state x_{k+1} and reward r_{k+1}
    $Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) +$
        $\alpha_k[r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u') - Q_k(x_k, u_k)]$
end for

Along the lines of Q-learning, one can derive an online RL algorithm for evaluating a given policy $\tilde{\pi}$:

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k[r_{k+1} + \gamma Q_k(x_{k+1}, u_{k+1}) - Q_k(x_k, u_k)] \quad (14)$$

assuming that actions are chosen with $\tilde{\pi}$, which is exploratory so that all actions are attempted (plus the standard learning rate conditions). Note the replacement of the maximizing action with the action actually chosen by the policy, which means the algorithm aims to solve the Bellman equation (6) for $\tilde{\pi}$ (or rather, an extended variant of (6) that also takes the expectation over actions according to $\tilde{\pi}$). This algorithm is called simply temporal difference, or TD, and we may imagine a model-free version of policy iteration (Algorithm 2) where TD runs in-between policy improvements for long enough intervals to allow Q-function convergence (a model-free version of Algorithm 3).

It turns out, however, that we may dispense with the requirement to converge before improvement, and in fact we may improve the policy at each step – implicitly, by selecting actions based on the current Q-function, e.g. with the ε-greedy or softmax policies (12), (13). Such a scheme is called optimistic policy improvement, and using it in combination to the update (14) leads to the SARSA algorithm (Rummery and Niranjan, 1994), named for the structure of the data tuple used by each update $(x_k, u_k, x_{k+1}, r_{k+1}, u_{k+1})$ – or state, action, reward, state, action. It is given in Algorithm 5. In order to converge to Q^*, SARSA requires the same conditions as Q-learning, and in addition that the exploratory policy being followed asymptotically becomes greedy, for instance by letting ε or τ go to 0 in ε-greedy or softmax (Singh et al., 2000). Note that unlike Q-learning, SARSA always aims to evaluate the policy that it follows, so it is on-policy.

Algorithm 5 SARSA.

initialize Q-function, e.g. $Q_0(x, u) = 0\ \forall x, u$
measure initial state x_0, choose arbitrary action u_0
for every time step $k = 0, 1, 2, \dots$ **do**
    apply u_k, measure next state x_{k+1} and reward r_{k+1}
    choose u_{k+1} with exploratory policy based on Q_k
    $Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) +$
        $\alpha_k[r_{k+1} + Q_k(x_{k+1}, u_{k+1}) - Q_k(x_k, u_k)]$
end for

The following two methods are often used to increase the convergence rate of temporal difference algorithms. The first method exploits the fact that the latest transition is the causal result of the entire preceding trajectory, by marking visited state-action pairs with eligibility traces $e : X \times U \to [0, \infty)$, see e.g. Singh and Sutton (1996). The traces are initially zero: $e_0(x, u) = 0\ \forall x, u$, and at each step the trace of the currently visited state-action pair is either set to 1 (in the replacing traces variant), or incremented by 1 (accumulating). For all other states, the trace decays with $\lambda\gamma$, where $\lambda \in [0, 1]$ is a parameter. Formally:

$$e_{k+1}(x_k, u_k) = \begin{cases} 1 & \text{if replacing} \\ e_k(x_k, u_k) + 1 & \text{if accumulating} \end{cases}$$
$$e_{k+1}(x, u) = \lambda\gamma e_k(x, u) \text{ for all } (x, u) \neq (x_k, u_k)$$

Note that state-action pairs become exponentially less eligible as they move further into the past. Consider e.g. the TD algorithm (14), and denote the temporal difference by $\delta_k = r_{k+1} + \gamma Q_k(x_{k+1}, u_{k+1}) - Q_k(x_k, u_k)$. The traces are used by updating at step k all eligible state-action pairs using this temporal difference, weighted by the traces:

$$Q_{k+1}(x, u) = Q_k(x, u) + \alpha_k \delta_k e_{k+1}(x, u),\ \forall x, u \quad (15)$$

So the latest transition sample is used to update many Q-values instead of one, increasing data efficiency. The algorithm obtained is called TD(λ), and it reduces to the original one when $\lambda = 0$. The same idea can be applied in a straightforward fashion to SARSA and Q-learning. Although for simplicity we introduced traces heuristically here, they have a deep analytical justification in terms of using longer-horizon updates than just the single-step rewards (Sutton and Barto, 2018).

The second method is called experience replay (Lin, 1992). Rather than using each transition only once, when it is observed, all transitions are saved in memory and "replayed", i.e. the original learning update is applied to them as if they would have been observed again. Two choices drive experience replay: how many stored transitions are replayed at each true step k; and how transitions are selected (e.g. randomly, forward or backward along trajectories). For the classical algorithms in this section, it is better to replay trajectories backwards, as this better propagates reward information.

3. Approximate reinforcement learning

3.1. Approximate representations

The methods in Section 2 require that Q-functions and policies are exactly represented – e.g. as a table indexed by the discrete states and actions. In typical control problems, states and actions are continuous, exact representations are in general impossible, and function approximation must be used.

Depending on the algorithm, the Q-function and/or the policy must be approximated, and we denote their approximate versions by $\hat{Q}$ and $\hat{\pi}$. Often, parametric approximators are used, in which case the Q-function parameters are denoted by $\theta \in \mathbb{R}^p$, and approximate Q-values by $\hat{Q}(x, u; \theta)$. For the policy, the parameters are $w \in \mathbb{R}^m$. Linearly parameterized architectures are

popular since they lead to relatively simple algorithms and analysis. For the Q-function, such a parametrization is written:

$$\hat{Q}(x,u;\theta) = \sum_{i=1}^{p} \phi_i(x,u)\theta_i = \phi^\top(x,u)\theta \tag{16}$$

where $\phi(x,u) = [\phi_1(x,u),\ldots,\phi_p(x,u)]^\top$ is a vector of basis functions (BFS), such as radial BFs or polynomial terms. A particularly simple linear architecture is aggregation, where the BFs are binary and equal to 1 in disjoint areas of the state-action space; thus every point in such an area has the same Q-value.

One example of nonlinear parametrization is a BF expansion in which the BF shapes themselves are also parameterized (Bertsekas and Yu, 2009), e.g. by the centers and radii for radial BFs. However, neural networks are probably the most widely used type of nonlinear approximator (Bertsekas and Tsitsiklis, 1996; Riedmiller, 2005), with deep networks becoming very popular recently (Mnih et al., 2015). The latter directly take images as their state input, and consist of many layers with specific structures; they are the focus of Section 5 later on. In the adaptive dynamic programming field (Lewis and Liu, 2012), approximators are also traditionally called neural networks, but usually the networks have one linear layer (or only the last linear layer is adapted), so in fact they boil down to linear approximators.

Many RL techniques also use nonparametric approximators, which vary their shape and number of parameters as a function of the dataset, e.g. RL with kernel regression (Ormoneit and Sen, 2002; Farahmand et al., 2009), Gaussian processes (Engel et al., 2005), regression trees (Ernst et al., 2005), etc.

The main advantage of function approximation is the reduction of the (generally intractable) problem of learning the continuous-argument Q-function or policy to the tractable problem of learning a parameter vector. There is also a less obvious benefit: as each parameter contributes to estimating the value of many state-action pairs, any transition from these pairs will help learning that parameter. This reduces the number of samples needed to learn, and is called generalization.

In the sequel, we describe some major approximate RL methods, see also Sutton and Barto (2018); Bertsekas (2012); Powell (2012); Buşoniu et al. (2010) for extensive reviews of the area. We start with offline methods in Section 3.2, and online temporal-difference RL in Section 3.3. The techniques we discuss in these sections assume the Q-function approximator is such that greedy actions can be efficiently found, see (5), and use this fact to sidestep the requirement of representing policies. Instead they simply find greedy actions on demand, at any state where they are needed. The usual way to achieve this is to discretize the actions into a few values, and then maximize by enumeration. For linear approximation (16), this can be imagined as removing the u parameter from the BFs and instead replicating state-dependent BFs for each discrete action, with different associated parameters. More flexible solutions exist, e.g. differentiating polynomial approximations to find the maxima, or performing binary search in the action space (Pazis and Lagoudakis, 2009). In Section 3.4, we discuss policy gradient techniques, which explicitly search over parameterized policies and do not need to maximize over actions.

3.2. Offline approximate RL

We describe two popular offline algorithms from the value and policy iteration class, respectively. While in Section 2.2 these methods were model-based, their extensions here use a given dataset of transition samples, so they become model-free RL when the samples are obtained from the system. Of course, if a model is in fact available, it can be used to generate the samples. The dataset is denoted $\mathcal{S} = \{(x_j, u_j, x'_j, r_j) | j = 1,\ldots,S\}$, where x'_j is sampled from $f(x_j,u_j,\cdot)$ and r_j is the corresponding reward. The dataset should be sufficiently informative to allow finding a good solution – an exploration requirement.

The first algorithm is fitted Q-iteration, and was popularized under this name by Ernst et al. (2005), although its basic principle was already known. It is applicable to any type of Q-function approximator, although for clarity we will use a parametric one $\hat{Q}(x,u;\theta)$. At iteration ℓ, when the parameters are θ_ℓ, fitted Q-iteration computes the Bellman target $q_{\ell+1,j} = r_j + \gamma \max_{u'} \hat{Q}(x'_j,u';\theta_\ell)$ for each transition sample. Then, least-squares regression is run on the input-output samples $(x_j,u_j) \mapsto q_{\ell+1,j}$ to obtain the next parameters $\theta_{\ell+1}$. Algorithm 6 summarizes the procedure. The target Q-value is exact in the deterministic case, when the Bellman equation is (8), and it is a sample of the right-hand side of (7) in the stochastic case. In the latter case, least-squares regression should approximate the correct expected value.

Algorithm 6 Fitted Q-iteration.

Input: γ, dataset $\mathcal{S}$

1: initialize parameter vector, e.g. $\theta_0 \leftarrow 0$
2: **repeat** at every iteration $\ell = 0,1,2,\ldots$
3: $\quad q_{\ell+1,j} = r_j + \gamma \max_{u'} \hat{Q}(x'_j,u';\theta_\ell)$, for $j = 1,\ldots,S$
4: $\quad \theta_{\ell+1} = \arg\min_\theta \sum_{j=1}^{S} \left[q_{\ell+1,j} - \hat{Q}(x_j,u_j;\theta)\right]^2$
5: **until** $\theta_{\ell+1}$ is satisfactory

Output: $\hat{Q}^*$, $\hat{\pi}^*$ greedy in $\hat{Q}^*$ (implicitly represented via $\hat{Q}^*$)

An asymptotic guarantee can be provided for fitted Q-iteration by first defining an error ε such that at any iteration ℓ, $\left\|T[\hat{Q}(\cdot,\cdot;\theta_\ell)] - \hat{Q}(\cdot,\cdot;\theta_{\ell+1})\right\|_\infty \leq \varepsilon$. Thus ε characterizes the worst-case error between the exact Q-functions that the Bellman mapping T would compute, and the actual, approximate Q-functions found by the algorithm. Then, fitted Q-iteration asymptotically reaches a sequence of approximate Q-functions that each satisfy $\left\|\hat{Q} - Q^*\right\|_\infty \leq \frac{\varepsilon}{1-\gamma}$ (Bertsekas and Tsitsiklis, 1996). While this bound holds for the Q-function and not the greedy policy, the following general result makes the connection. If an approximately optimal Q-function $\hat{Q}^*$ is available, then a greedy policy $\hat{\pi}^*$ in this Q-function satisfies $\left\|Q^{\hat{\pi}^*} - Q^*\right\|_\infty \leq \frac{2\gamma}{1-\gamma}\left\|\hat{Q}^* - Q^*\right\|_\infty$. Thus, overall, fitted Q-iteration asymptotically satisfies $\left\|Q^{\hat{\pi}^*} - Q^*\right\|_\infty \leq \frac{2\gamma\varepsilon}{(1-\gamma)^2}$. So-called finite-sample guarantees, which work for other norms and make explicit the dependence on the number of samples and iterations, are given by Munos and Szepesvári (2008).

Fitted Q-iteration may not converge to some fixed Q-function. Under stronger conditions, including that the approximator does not extrapolate sample values, fitted Q-iteration updates are contractive and converge to a fixed point (Gordon, 1995; Ormoneit and Sen, 2002). A simple such case satisfying the condition is interpolation on a grid, with the samples equal to the grid points.

Consider next policy iteration, and recall from Section 3.1 that policy approximation is sidestepped by computing greedy actions on demand. Thus the core feature of approximate policy iteration is the policy evaluation procedure. A fitted algorithm like for Q-iteration may be used, but it turns out that if the approximator is linear (16), more efficient procedures can be devised by exploiting the linearity of the Bellman equation (6). One such procedure is least-squares temporal difference (LSTD), introduced for value functions V by Bradtke and Barto (1996). When applied to find Q-functions, LSTD leads to least-squares policy iteration (LSPI) (Lagoudakis and Parr, 2003). We return temporarily to discrete spaces, since the derivation is better understood there, but the final method applies to the continuous case as well. Let the discrete states and actions be denoted by their index $x = 1, \ldots, N$ and $u = 1, \ldots, M$. We start by rewriting the Bellman mapping T^π, see again (6):

$$T^\pi[Q](x,u) = \bar{\rho}(x,u) + \gamma \sum_{x'=1}^{N} f(x,u,x') Q(x', \pi(x')) \qquad (17)$$

where $\bar{\rho}(x,u) = \sum_{x'=1}^{N} f(x,u,x')\rho(x,u,x')$ are the expected rewards. Now, since the solution to the Bellman equation $Q^\pi = T^\pi[Q^\pi]$ is generally not representable by the chosen approximator, the idea in LSTD is to solve a projected version $\hat{Q} = P[T^\pi[\hat{Q}]]$, in which the result of T^π is brought back into the space of representable Q-functions via a weighted least-squares projection $P[Q] = \arg\min_{\hat{Q}} \sum_{x,u} d(x,u) \left| Q(x,u) - \hat{Q}(x,u) \right|^2$. Here, d gives the weights of state-action pairs.

To exploit linearity, define now a vector form of the Q-function, $\boldsymbol{Q} \in \mathbb{R}^{NM}$, in which $\boldsymbol{Q}_{\underline{xu}} = Q(x,u)$ for scalar integer index $\underline{xu} = x + (u-1)N$. Then, (17) can be rewritten in matrix form:

$$\boldsymbol{T}^\pi[\boldsymbol{Q}] = \boldsymbol{\rho} + \gamma \boldsymbol{f} \boldsymbol{Q} \qquad (18)$$

where $\boldsymbol{\rho}$ collects expected rewards in a similar way to $\boldsymbol{Q}$, and $\boldsymbol{f} \in \mathbb{R}^{NM \times NM}$ is a matrix of transition probabilities between state-action pairs. For a deterministic policy, $\boldsymbol{f}_{\underline{xu},\underline{x'u'}} = f(x,u,x')$ if $u' = \pi(x')$, and 0 elsewhere, but the formalism generalizes to stochastic, exploratory policies.

Further, by collecting the basis function values in a matrix $\Phi \in \mathbb{R}^{NM \times p}$, $\Phi_{\underline{xu},i} = \phi_i(x,u)$, an approximate Q-function is written $\hat{\boldsymbol{Q}} = \Phi\theta$. Weighted least-squares projection can be written in closed form as $\boldsymbol{P} = \Phi(\Phi^\top \boldsymbol{d} \Phi)^{-1} \Phi^\top \boldsymbol{d}$, where $\boldsymbol{d}$ is the weight vector, with $\boldsymbol{d}_{\underline{xu}} = d(x,u)$. Finally, replacing the matrix forms $\hat{\boldsymbol{Q}}$ and $\boldsymbol{P}$ into (18), the projected Bellman equation becomes after a few manipulations:

$$(\Phi^\top \boldsymbol{d} \Phi - \gamma \Phi^\top \boldsymbol{d} \boldsymbol{f} \Phi)\theta = \Phi^\top \boldsymbol{d} \boldsymbol{\rho}, \text{ or equivalently } A\theta = b$$

with $A := \Phi^\top \boldsymbol{d} \Phi - \gamma \Phi^\top \boldsymbol{d} \boldsymbol{f} \Phi \in \mathbb{R}^{p \times p}$, and $b := \Phi^\top \boldsymbol{d} \boldsymbol{\rho} \in \mathbb{R}^p$. While this equation involves the model, A and b can fortunately be estimated in a model-free fashion, from samples drawn according to the weights d (reinterpreted as probabilities). The formulas are given directly in Algorithm 7, where lines 3–5 comprise LSTD. Recall that the algorithm works for continuous variables as well.

Algorithm 7 Least-squares policy iteration.

Input: γ, dataset $\mathcal{S}$

1: initialize policy π_0
2: **repeat** at every iteration $\ell = 0, 1, 2, \ldots$
3: $\quad A_\ell = \sum_{j=1}^{S} \phi(x_j, u_j) \left[\phi^\top(x_j, u_j) - \gamma \phi^\top(x'_j, \pi_\ell(x'_j)) \right]$
4: $\quad b_\ell = \sum_{j=1}^{S} \phi(x_j, u_j) r_j$
5: $\quad$ solve $A_\ell \theta_\ell = b_\ell$ to get Q-function parameters θ_ℓ
6: $\quad \pi_{\ell+1}(x) = \arg\max_u \hat{Q}(x, u; \theta_\ell), \forall x$ (implicitly)
7: **until** $\pi_{\ell+1}$ is satisfactory

Output: $\hat{\pi}^* = \pi_{\ell+1}$

The asymptotic properties of LSPI, and more generally of approximate policy iteration, are similar to those of fitted Q-iteration. In particular, the algorithm may never reach a fixed point, but if at each iteration the policy evaluation error $\left\| \hat{Q}(x,u;\theta_\ell) - Q^{\pi_\ell} \right\|_\infty$ is upper-bounded by ε, then the algorithm eventually reaches a sequence of policies such that $\|Q^\pi - Q^*\|_\infty \leq \frac{2\gamma\varepsilon}{(1-\gamma)^2}$ (Lagoudakis and Parr, 2003). The policy evaluation error can be specifically characterized for LSTD, see Lazaric et al. (2012) for a finite-sample analysis of LSTD and LSPI, and Buşoniu et al. (2011) for a review. In practice, when the algorithms converge, LSPI often needs fewer iterations than fitted Q-iteration, mirroring a similar relationship for exact policy and value iteration. An alternative to LSTD is an iterative algorithm called LS policy evaluation, analyzed by Yu and Bertsekas (2009); while instead of solving the projected equation we may also minimize the difference between the two sides of (6), leading to Bellman residual minimization, see Scherrer (2010) for a comparison between this and the projection-based algorithms.

Research on batch RL methods is ongoing, with recent developments in e.g. nonparametric approximation (Farahmand et al., 2016), exploitation of low-level controllers called "options" (Mann et al., 2015), and combinations with deep learning (Lee et al., 2017). Combinations of this latter type are presented in detail in the upcoming Section 5, although we already note that the usage of so-called minibatches of samples in deep RL is closely connected to fitted Q-iteration.

3.3. Online, temporal-difference approximate RL

To derive a simple variant of approximate Q-learning, we will reinterpret the term $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1}, u')$ in the original Q-learning (11) as a target of the incremental update. In the approximate case, we replace this target by $q_{k+1} = r_{k+1} + \gamma \max_{u'} \hat{Q}(x_{k+1}, u'; \theta_k)$ which has the same form as the Bellman target from fitted Q-iteration, except that now instead of an offline dataset we use samples observed online, and update the

parameters at each step k. The main idea is to perform gradient descent on the squared error between this target and the approximate Q-value (Sutton and Barto, 1998, Ch. 8):

$$\begin{aligned}\theta_{k+1} &= \theta_k - \alpha_k \nabla \left(\frac{1}{2} [q_{k+1} - \hat{Q}(x_k, u_k; \theta_k)]^2 \right) \\ &= \theta_k + \alpha_k [q_{k+1} - \hat{Q}(x_k, u_k; \theta_k)] \nabla \hat{Q}(x_k, u_k; \theta_k) \\ &= \theta_k + \alpha_k [r_{k+1} + \gamma \max_{u'} \hat{Q}(x_{k+1}, u'; \theta_k) \\ &\qquad - \hat{Q}(x_k, u_k; \theta_k)] \nabla \hat{Q}(x_k, u_k; \theta_k)\end{aligned} \tag{19}$$

where the target was held constant in the first two equalities, the gradient is always with respect to the parameters, and notation $\nabla g(\theta_k)$ means that the gradient of g is evaluated at point θ_k. Note that an approximate temporal difference $\hat{\delta}_k$ has been obtained in the square brackets. For linear parameterizations (16), the derivative of $Q(x_k, u_k; \theta)$ is very simple: it is equal to the vector of basis functions $\phi(x_k, u_k)$. We do not provide explicit pseudocode of the algorithm, since it is easily obtained by replacing the exact update in Algorithm 4 by (19).

An approximate SARSA variant is similarly derived:

$$\begin{aligned}\theta_{k+1} = \theta_k + \alpha_k [r_{k+1} + \gamma \hat{Q}(x_{k+1}, u_{k+1}; \theta_k) \\ - \hat{Q}(x_k, u_k; \theta_k)] \nabla \hat{Q}(x_k, u_k; \theta_k)\end{aligned}$$

where the policy is based on $\hat{Q}$. If the policy is held constant, then this update performs policy evaluation, and it becomes a gradient-based, approximate version of the TD method.

Just like in the exact case, exploration is required by these approximate online methods, and the learning rate and exploration schedules are essential for learning speed. Experience replay extends in the obvious way: by reapplying the gradient-based updates to the stored samples. Eligibility traces $e \in \mathbb{R}^p$ now accumulate the impact of the parameters on the updates, as measured by the gradients: $e_{k+1} = \gamma \lambda e_k + \nabla \hat{Q}(x_k, u_k; \theta_k)$, where the traces are initialized at zero. Then, the updates are changed so that the approximate temporal difference is weighted by the eligibility trace instead of just the latest gradient: $\theta_{k+1} = \theta_k + \hat{\delta}_k e_{k+1}$. By plugging in the appropriate $\hat{\delta}_k$, this procedure applies equally well to approximate SARSA, Q-learning, and TD. For clarity, Algorithm 8 exemplifies the complete gradient-based SARSA(λ) method. Note that LSTD and other methods in the least-squares family can also be extended to use eligibility traces (Thiery and Scherrer, 2010).

Algorithm 8 Approximate SARSA(λ).

1: initialize parameters, e.g. $\theta_0 = 0_p$, and traces $e_0 = 0_p$
2: measure initial state x_0, choose arbitrary action u_0
3: **for** every time step $k = 0, 1, 2, \ldots$ **do**
4: apply u_k, measure next state x_{k+1} and reward r_{k+1}
5: choose u_{k+1} with explor. policy based on $\hat{Q}(x_{k+1}, \cdot, \theta_k)$
6: $\hat{\delta}_k = r_{k+1} + \gamma \hat{Q}(x_{k+1}, u_{k+1}; \theta_k) - \hat{Q}(x_k, u_k; \theta_k)$
7: $e_{k+1} = \gamma \lambda e_k + \nabla \hat{Q}(x_k, u_k; \theta_k)$
8: $\theta_{k+1} = \theta_k + \hat{\delta}_k e_{k+1}$
9: **end for**

While the derivations above are heuristic, the overall idea is sound. Tsitsiklis and Van Roy (1997) have analyzed approximate TD, while a comparison between gradient-based TD and LSTD from Section 3.2 is provided by Yu and Bertsekas (2009). Convergence of approximate Q-learning has been proven for linear approximators, initially under the restrictive requirement that the policy followed is constant (Melo et al., 2008). This is of course unrealistic, and finding good off-policy, online approximate RL methods has been the focus of many research efforts, see e.g. Munos et al. (2016); Sutton et al. (2016). Most of the effort in online RL in the last few years has however been focused on deep approximation, and we postpone that discussion until the dedicated Section 5.

3.4. Policy gradient and actor-critic methods

The approximate RL algorithms reviewed so far require fast maximization of Q-functions over actions, which usually means the actions are discretized, as explained in Section 3.1. Exploiting this, policies are represented implicitly, via the Q-functions. Both of these features may sometimes be undesirable in control problems. To accurately stabilize to some state or track some trajectory, continuous actions are generally needed. Furthermore, some prior knowledge on the policy may be available, which may be included in a policy parametrization (e.g. the policy may be initialized to a linear controller that is known to work locally), whereas translating such prior knowledge into an initial shape of the Q-function is highly nontrivial.

Policy gradient techniques solve these issues by choosing to represent the policy explicitly, almost always with a parametric approximator and including an exploration term. The policy is then $\hat{\pi}(x, u; w)$, where for each $x \in X$ and $w \in \mathbb{R}^m$, $\hat{\pi}(x, \cdot; w)$ is a probability density over the continuous action space U. Often, this density is a Gaussian centered on some parameterized deterministic action. Below we review some major ideas in policy gradients, see Deisenroth et al. (2011); Grondman et al. (2012) for dedicated reviews of the area.

As hinted by the name, policy gradient methods perform gradient ascent on a scalar objective function, defined as the expected return when drawing the initial state from a distribution $d_0(x)$ (while the actions and next states follow their own distributions):

$$\begin{aligned}J(w) &= E_{\substack{x_0 \sim d_0(\cdot),\ u_k \sim \hat{\pi}(x_k, \cdot; w), \\ x_{k+1} \sim f(x_k, u_k, \cdot)}} \left\{ \sum_{k=0}^{\infty} \gamma^k \rho(x_k, u_k, x_{k+1}) \right\} \\ &= \int_X d^w(x) \int_U \hat{\pi}(x, u; w) \int_X f(x, u, x') \rho(x, u, x') \mathrm{d}x' \mathrm{d}u \mathrm{d}x\end{aligned}$$

The second formula rewrites the expectation using the so-called discounted state distribution $d^w(x) = \sum_{k=0}^{\infty} \gamma^k \mathrm{P}(x_k = x)$, which sums up the probabilities of encountering that state at each subsequent step, properly discounted for that step (Sutton et al., 2000). This distribution is superscripted by w as it depends (through the policy) on the parameters. By changing d^w, the discounted returns to be optimized can be replaced by average rewards over time, and many policy gradient methods are originally given for such average rewards. Note that by choosing d_0, we may focus the algorithm on interesting initial state regions, or even just a few specific states.

The core gradient ascent update is then written:

$$w_{\ell+1} = w_\ell + \alpha_\ell \nabla J(w_\ell)$$

where ℓ is the iteration, and the learning rates α_ℓ must obey the usual stochastic approximation conditions (i)-(ii) from Section 2.3. The key question is how to estimate the gradient in this formula. Many methods use Monte-Carlo estimation with trajectories sampled using the current policy, often called roll-outs. For instance, REINFORCE (Williams, 1992) and GPOMDP (Baxter and Bartlett, 2001) are two classical such methods. Such direct estimation methods are very general, but may suffer from large variance of the gradient estimates and therefore slow learning.

Actor-critic methods tackle this problem by using a value function (the critic) to compute the gradient of the policy (the actor). The fundamental connection between these two quantities is given by the policy gradient theorem, discovered simultaneously by Sutton et al. (2000) and by Konda and Tsitsiklis (2003). When applied in the approximate case, this theorem requires that the Q-function is represented using a so-called compatible approximator, which is linear and uses the BFs $\phi(x,u) = \nabla \log \hat{\pi}(x,u;w)$:

$$\hat{Q}(x,u;\theta) = [\nabla \log \hat{\pi}(x,u;w)]^\top \theta$$

Then, the policy gradient theorem states that:

$$\nabla J(w) = \int_X d^w(x) \int_U \nabla \hat{\pi}(x,u;w) \hat{Q}(x,u;\theta) \mathrm{d}u \mathrm{d}x \tag{20}$$

assuming that θ has been found so that $\hat{Q}(x,u;\theta)$ is a least-squares approximation of the true Q-function of the policy $\hat{\pi}(x,u;w)$ given by the current parameter w. Compatible approximation provides a major advantage: once a good policy parametrization has been found (from prior knowledge or otherwise), a Q-function parametrization automatically follows. Intuitively, (20) says that the least-squares projection of the Q-function on the span of the compatible BFs provides sufficient information to compute the gradient. To find θ, approximate policy evaluation may be performed e.g. with the TD or LSTD techniques above.

A few landmark references about actor-critic methods include Barto et al. (1983), where the actor-critic structure was first defined; Peters and Schaal (2008) which popularized the so-called natural actor critic; and Bhatnagar et al. (2009) where an array of such algorithms with convergence proofs was given. We outline here the particularly elegant method of Peters and Schaal (2008). It relies on the natural gradient (Amari and Douglas, 1998; Kakade, 2001), which rescales the gradient $\nabla J(w)$ by the inverse of the curvature, somewhat like Newton's method for optimization but without assuming a locally quadratic shape. The details are involved, but the final result is that for a compatible Q-function approximator, the natural gradient is equal the Q-function parameters θ, leading to the simple update:

$$w_{\ell+1} = w_\ell + \alpha_\ell \theta_\ell$$

4. Control-theoretic approaches and viewpoint

Control-theoretic approaches and techniques for approximately solving optimal control problems of the type (2)-(3) are labeled Approximate or Adaptive Dynamic Programming (ADP). Many of these approaches are model-free, in which case they are classified as RL. However, not all ADP methods involve learning or have a direct connection with RL.

Owing to the broadness of the terms ADP and RL, many authors utilize them interchangeably, and ADP is sometimes also used to denote AI approaches. To keep the terminology consistent, in this survey we reserve the term ADP only for control-theoretic approaches; and RL only for model-free methods – regardless of whether they originate in AI or control.

In contrast to the other methods in this paper, control-theoretic approaches often rely on state-dependent value functions V^π, V^*, so we will characterize them briefly here. The Bellman equations for the policy and optimal value functions are, respectively:

$$V^\pi(x) = E_{x' \sim f(x,\pi(x),\cdot)} \{\rho(x,\pi(x),x') + \gamma V^\pi(x')\} \tag{21}$$

$$V^*(x) = \max_u E_{x' \sim f(x,u,\cdot)} \{\rho(x,u,x') + \gamma V^*(x')\} \tag{22}$$

Similarly to $T[Q]$, we will also interpret the right-hand side of (22) as a mapping applied to the value function, denoted $\mathcal{T}$, so that $V^* = \mathcal{T}[V^*]$. A temporal-difference error similar to the one in square brackets in (14), but written in terms of value function V, is:

$$r_{k+1} + \gamma V(x_{k+1}) - V(x_k) \tag{23}$$

In the remainder of this section, we first reflect on RL stability considerations (Section 4.1), and then in Section 4.2 we present ADP based on Linear Programming (LP).

4.1. Stability considerations for reinforcement learning

As previously stated, (2)-(3) is an optimal control problem for sequential decision making under uncertainties (Powell, 2012; Russell and Norvig, 2016; Sutton and Barto, 2018; Bertsekas, 2017; Borrelli et al., 2017). Optimal control methods typically seek control laws in an offline manner assuming availability of the underlying dynamic models (Liberzon, 2011; Borrelli et al., 2017). When the underlying models are unavailable or partially known, adaptive control approaches are employed in an online fashion (Landau et al., 2011). Thus, online RL methods (e.g., Section 2.3) represent adaptive optimal control methods in the sense that (sub)optimal control laws are obtained online using real-time measurements without a model (Lewis et al., 2012; Sutton and Barto, 2018).

Stability analyses of optimal and adaptive control methods are crucial in safety-related and potentially hazardous applications such as human-robot interaction, autonomous robotics or power plant control. In order to have tractable and conclusive solutions, the control community often starts from deterministic settings when devising optimal and adaptive control laws. Subsequently, potential uncertainties, noise and disturbances, which are not found in the deterministic setting, are handled using various robustness tools (e.g., dissipativity, Lp-stability,

Input-to-State Stability, etc.) and simplifications (e.g., certainty equivalence) in order to infer stability in an appropriate sense, such as local, set, asymptotic, exponential or semi-global stability, as well as uniform ultimate boundedness (Zhang et al., 2011; Lewis et al., 2012; Yang et al., 2014; Görges, 2017; Postoyan et al., 2017).

Unlike the standard control approaches (Liberzon, 2011; Landau et al., 2011; Borrelli et al., 2017), which are designed around stability from the start, RL approaches necessitate further stability, feasibility and robustness guarantees (Lewis et al., 2012; Görges, 2017). It is worth pointing out that here we are interested in stability from the control viewpoint, i.e., stability of the closed-loop system resulting from using the (e.g. online learning) controller. This should not be confused with another notion of stability often used in AI, which refers to convergence of learning algorithms (to the asymptotic behavior from the control viewpoint). We always use in this review the word "convergence" for the latter notion.

This highlights a fundamental philosophical difference between the AI and control communities. AI researchers focus on performance in terms of cumulative rewards, where the rewards can have any meaning and are seen as a given part of the problem. Algorithmically, this means that only the convergence (qualitative/asymptotic, or quantitative via convergence rates) of the learning process to a near-optimal solution is considered, whilst overshoot bounds along the learning process (that is, the so-called δ-ϵ arguments), which are needed for closed-loop stability, are put aside. This is sometimes possible due to innocuous nature of some AI applications (e.g., mastering video or board games), while for physical systems such as robots stability is resolved heuristically. Conversely, the objectives of control researchers revolve around stability, so that even when optimal control is used, the major – and often, the only – role of the rewards (equivalently, negative costs) is to represent stability requirements, such as in standard approaches to Model Predictive Control (MPC). There exist of course exceptions – for instance, economic MPC methods (Diehl et al., 2011) reintroduce "true" optimization objectives in MPC.

This basic difference leads to other, subtler variations between the fields. For example, in AI rewards are often assumed to be bounded from the outset, whereas in control they are not because if the state unstably grows arbitrarily large this should be reflected by arbitrary (negative) reward magnitudes. Similarly, a discount factor is not typically used in control because the value function is no longer a Lyapunov function, so that optimal solutions of (2)-(3) might not be stable at all (Postoyan et al., 2017). Furthermore, since without some knowledge about the system it is impossible to provide an (at least initial) stabilizing controller, control approaches typically assume some model knowledge for control design and then add uncertainty on top of this basic model, as explained earlier. On the other hand, AI usually takes the view that nothing at all is known about the dynamics, which – in addition to making stability guarantees very difficult – also leads to a tendency in AI to ignore any existing knowledge about the model, much to the puzzlement of the control engineer. We provide in Section 6 some ideas on how these two different overall views might be reconciled.

Next, to help ground things for the control engineer, let us notice that closed-form solutions of (2)-(3) are available in some specific cases: Linear Quadratic Regulation (LQR) problems, which were presented in Section 2.1, Linear Quadratic Gaussian (LQG) problems (Bertsekas, 2017; Powell, 2012; Lewis et al., 2012; Modares et al., 2015; Sutton and Barto, 2018), and H_∞-control. Even though the H_∞, LQR and LQG (iterative) solutions are not originally derived using the RL framework, these solutions (e.g., the value-function-based Hewer's and Lyapunov recursion algorithms) are readily derived using the RL approaches presented in previous sections as shown by Bertsekas (2017); Powell (2012); Lewis et al. (2012); Sutton and Barto (2018).

We delve deeper into the standard LQR problem. When $\gamma = 1$, the following Hamiltonian function (Lewis et al., 2012):

$$\begin{aligned} H(x_k, u_k) =& x_k^\top \mathbb{Q} x_k + u_k^\top \mathbb{R} u_k + (Ax_k + Bu_k)^\top \mathbb{P}(Ax_k + Bu_k) \\ & - x_k^\top \mathbb{P} x_k, \end{aligned}$$

where $\mathbb{P}$ is the Riccati solution, is in fact a temporal difference error (23). This is because $x_k^\top \mathbb{Q} x_k + u_k^\top \mathbb{R} u_k = r_{k+1}$, $(Ax_k + Bu_k)^\top \mathbb{P}(Ax_k + Bu_k) = V^*(x_{k+1})$, and $x_k^\top \mathbb{P} x_k = V^*(x_k)$. A necessary condition for optimality $\partial H(x_k, u_k)/\partial u_k = 0$ yields the stabilizing and optimal control law:

$$u_k = \pi^*(x_k) = -(B^\top \mathbb{P} B + \mathbb{R})^{-1} B^\top \mathbb{P} A x_k = L x_k, \tag{24}$$

which is the undiscounted version of (10). Next, since it also holds that:

$$V^*(x_k) = \sum_{i=k}^{\infty} r_{i+1} = x_k^\top \mathbb{Q} x_k + u_k^\top \mathbb{R} u_k + V^*(x_{k+1}),$$

we have that:

$$x_k^\top \mathbb{P} x_k = x_k^\top \mathbb{Q} x_k + x_k^\top L \mathbb{R} L^\top x_k + x_k^\top (A + BL)^\top \mathbb{P}(A + BL) x_k$$

yielding:

$$\begin{aligned} \mathbb{P} &= \mathbb{Q} + L \mathbb{R} L^\top + (A + BL)^\top \mathbb{P}(A + BL), \\ &= A^\top [\mathbb{P} - \mathbb{P} B (\mathbb{R} + B^\top \mathbb{P} B)^{-1} B^\top \mathbb{P}] A + \mathbb{Q} \end{aligned}$$

which is the undiscounted version of the Riccati equation (9).

To apply RL, we will use Q-functions. In the LQR case, the optimal Q-function is:

$$Q^*(x_k, u_k) = x_k^\top \mathbb{Q} x_k + u_k^\top \mathbb{R} u_k + (Ax_k + Bu_k)^\top \mathbb{P}(Ax_k + Bu_k)$$

This can be rewritten in the following quadratic form (Lewis and Vrabie, 2009):

$$Q^*(x_k, u_k) = \frac{1}{2} \begin{bmatrix} x_k \\ u_k \end{bmatrix}^\top \underbrace{\begin{bmatrix} A^\top \mathbb{P} A + \mathbb{Q} & B^\top \mathbb{P} A \\ A^\top \mathbb{P} B & B^\top \mathbb{P} B + \mathbb{R} \end{bmatrix}}_{=: \begin{bmatrix} Q_{xx} & Q_{xu} \\ Q_{ux} & Q_{uu} \end{bmatrix}} \begin{bmatrix} x_k \\ u_k \end{bmatrix} \tag{25}$$

By solving then $\partial Q^*(x_k, u_k)/\partial u_k = 0$, we get:

$$u_k = Q_{uu}^{-1} Q_{ux} x_k, \tag{26}$$

which is in fact (24). Nevertheless, although both (24) and (26) render the optimal control law, these two expressions are conceptually different. If the system model (i.e., the matrices A and B) is known, one can readily utilize (10). However, in RL the model is unknown, in which case (26) has a crucial advantage: it can be learned from data. This is done by learning the matrix of Q-function parameters in the form (25), which renders the optimal LQR control law u_k via (26). In other words, RL provides means to solve the algebraic Riccati equation (9) without knowing the model, possibly in an online manner using data measured along the system trajectories. Note that exploration is important in such a learning procedure; in control-theoretic approaches, exploratory actions are often obtained by adding so-called probing noise to the control signals – similarly to the policy gradient methods of Section 3.4.

For deterministic control-affine plants:

$$x_{k+1} = \bar{f}_x(x_k) + \bar{f}_u(x_k)u_k, \tag{27}$$

similar conclusions can be obtained even without the knowledge of $\bar{f}_x$ and $\bar{f}_u$, as demonstrated by Zhang et al. (2011); Lewis et al. (2012); Yang et al. (2014).

This line of deriving algorithms leads to the ADP field, which has stability as its primary focus, and most of the papers cited above are from ADP. However, the stability guarantees for H_∞, LQR and LQG are not given in the realm of RL, but by referring to the original (model-based) works. These works typically consider offline algorithms with perfect model knowledge and state information so that closed-loop stability during learning is not a relevant concern. Somewhat reversed approaches, in which the authors start off with robust stability (e.g., Lp-stability) and employ RL towards (sub)optimality, are also of interest (Kretchmar et al., 2001; Anderson et al., 2007; Friedrich and Buss, 2017; Tolić and Palunko, 2017). Such approaches follow the control philosophy of trading off optimality for stability.

A general framework for investigating stability of (sub)optimal control laws for deterministic discounted problems was given by Postoyan et al. (2017). Even though Postoyan et al. (2017) do not deal with solving (2)-(3), which is the focus of this paper, they elucidate connections between control-theoretic and AI approaches towards solving the deterministic version of (2)-(3). The main insight is that discounting, which is preferred in AI because it leads to nice fixed-point properties and algorithm convergence, can lead to instability unless care is taken to select γ sufficiently close to 1. Note that the default choice in control is $\gamma = 1$, as we selected it in the equations above, but this can lead to unboundedness of solutions unless stability is addressed (which is typically not done in AI works). Postoyan et al. (2017) also highlight novel connections between MPC (Grimm et al., 2005; Grüne and Pannek, 2016; Borrelli et al., 2017) and RL. For further comparisons among MPC and RL, refer to (Bertsekas, 2005; Ernst et al., 2009; Beuchat et al., 2016; Görges, 2017) and references therein. Accordingly, MPC is model-based, not adaptive, with high online complexity, but with a mature stability, feasibility and robustness theory as well as inherent constraint handling. On the other hand, RL is model-free, adaptive, with low online complexity, but with immature stability, feasibility and robustness theory as well as difficult constraint handling. It appears that the synergy of MPC and RL is a promising research avenue for handling (2)-(3) with stability guarantees.

Lastly, because guaranteeing stability of RL is a formidable challenge, many existing results suffer from a number of shortcomings. Most algorithms for nonlinear systems rely on function approximation for the reasons explained in Section 3, and a rationale behind employing approximate architectures is the fact that basis functions can (uniformly) approximate any continuous functions with arbitrary precision on compact sets. However, what if the underlying optimal value function (i.e., Q- or V-function) is not continuous? Can this discontinuity be determined a priori for some problems so that the approximation architecture and stability analyses can be modified accordingly? In addition, during the learning process, how can one guarantee that this compact set will not be left, especially in stochastic settings? Notice that the approximation error outside the compact set may not be upper bounded. Which initial conditions yield trajectories within the compact set of interest? Furthermore, while convergence results resolve stability issues for offline learning to some extent (since RL does not run in closed loop with the system), how to ensure that all iterations yield stable control laws during *online* RL, especially in stochastic environments? Even for deterministic problems, some of these questions are still not fully addressed (Lewis et al., 2012). All the above approximation and stability considerations are even more difficult when considering continuous-time dynamics and rewards as exemplified by integral RL (Lewis et al., 2012) and related approaches (Jiang and Jiang, 2012).

4.2. Linear programming approaches

Not all ADP approaches for solving (2)-(3) have the RL flavor. The works of de Farias and Roy (2003); Wang et al. (2014); Beuchat et al. (2016) devise LP approaches to ADP by formulating linear optimization problems whose solution corresponds to the solution of the optimal control problem (2)-(3) for $\gamma \in (0, 1)$. The rationale is that LP is a fast, effective and well-understood optimization tool.

Although LP approaches have different viewpoints and strategies, they are built upon the following two properties:

- *monotonicity*: for functions $V, V' : X \to \mathbb{R}$ the following holds

$$V(x) \leq V'(x), \forall x \in X \implies \mathcal{T}[V] \leq \mathcal{T}[V'], \tag{28}$$

  where function inequalities are interpreted pointwise, and

- *value iteration convergence*: similarly to the Q-function case discussed in Section 2, for any bounded initial value function $V_0 : X \to \mathbb{R}$ and any $x \in X$, since $\gamma < 1$, the following holds

$$V^*(x) = \lim_{k\to\infty} \mathcal{T}^k[V_0](x), \tag{29}$$

Now, an LP counterpart of (2)-(3) is:

$$\begin{aligned} \text{maximize} \quad & E_{x_0 \sim d_0(\cdot)} \hat{V}(x) \\ \text{subject to} \quad & \hat{V} \leq \mathcal{T}[\hat{V}], \end{aligned} \tag{30}$$

where $\hat{V}(x) = \sum_{i=1}^{p} \phi_i(x)\theta_i$ is an underestimator of the fixed point V^* (also, $\theta_i \in \mathbb{R}$, $\phi_i : X \to \mathbb{R}$). Note that, just like in most methods discussed herein, function approximation (in this case with BFs) is employed.

In general, the constraint in (30) is known as the Bellman inequality and represents a relaxation of the Bellman equation (22). The Bellman inequality is not linear in $\hat{V}$ owing to the max operator in (22). Therefore, LP approaches seek linear conditions/constraints that imply $\hat{V} \leq \mathcal{T}[\hat{V}]$. For instance, (30) is a linear program in the case of finite state and input spaces (de Farias and Roy, 2003). In addition, $\hat{V} \leq \mathcal{T}[\hat{V}]$ in (30) is often replaced with the iterated Bellman inequalities, that is, $\hat{V} \leq \mathcal{T}^K[\hat{V}]$, $K > 1$, $K \in \mathbb{N}$, in an effort to obtain less conservative estimates of V^*. As in the case of the Bellman inequality, the iterated Bellman inequality is often replaced by conditions/constraints that imply it. Similar lines of reasoning apply when Q-functions, rather than V-functions, are of interest as exemplified by Beuchat et al. (2016).

Performance bounds similar in spirit to those presented in the earlier sections are obtained for LP approaches as well. In addition, online variants of (30) are devised by de Farias and Roy (2003); Wang et al. (2014); Beuchat et al. (2016).

5. Deep reinforcement learning

Next, we shift our focus closer to AI in order to discuss a new subfield of RL that is extremely promising and has consequently seen a surge of research effort in recent years. This field is Deep Reinforcement Learning (DRL), and can be understood as a particularly powerful way to solve function approximation in RL, as introduced in Section 3.1.

There are many different function approximators to choose from, and all make some assumptions about the functions that need to be approximated. Neural Networks (NNs) make only smoothness assumptions and, as a consequence, are able to represent any smooth function arbitrarily well given enough parameters (Hornik, 1991), making them a very general approximator option. However, without assumptions in addition to smoothness, it is impossible to learn to approximate certain complex functions in a statistically efficient manner (Bengio et al., 2006). The most important additional assumption made in Deep Neural Networks (DNNs) is that the function that needs to be approximated can be composed of a hierarchy of simpler functions (Goodfellow et al., 2016). This assumption is expressed through the architecture of DNNs, which have multiple hidden layers that compute nonlinear transformations of the outputs of previous layers. This decomposability assumption has proven very useful, especially when learning functions of natural data such as images, sounds and languages.

The combination of these DNN function approximators with RL into DRL is tempting, especially for domains such as robotics where it can enable learning behaviors directly from raw sensory signals through trial and error. DRL has already shown impressive results such as achieving super-human performance on the game of Go, which until recently was believed to require human intuition (Silver et al., 2016). It is however important to realize that the assumptions behind DNNs do not always hold and that they do come at a price. We outline the assumptions, the opportunities they offer and the potential pitfalls of combining DNNs with RL in Section 5.1. In Section 5.2, we describe common general strategies to deal with the challenges of DRL, while Section 5.3 gives an overview of popular DRL algorithms and how they implement the solutions. Section 5.4 describes ways in which the opportunities provided by the DNN assumptions can be exploited further.

5.1. Opportunities and pitfalls

In order to decide whether using a DNN as a function approximator is a good idea, and to realize the potential when one is used, it is important to be aware of the consequences stemming from the assumptions underlying deep learning.

Universal function approximation

The use of a universal function approximator, which can approximate any smooth function arbitrarily well, makes it possible to learn complex nonlinear policies and value functions. Theoretically, the combination of RL with DNNs gives a very general algorithm. However, this does mean that the space of possible functions is very large, making the optimization problem of finding a good set of parameters difficult. When more is known about the properties of the function that needs to be approximated, including this knowledge and thereby reducing the search space can be very beneficial. Although additional assumptions might introduce bias in the learned function, it might also make the problem of learning the function tractable. Additionally, the use of a universal function approximator makes it more likely to over-fit to the training data. Rajeswaran et al. (2017) showed how, on a set of benchmarks often used to test DRL algorithms, RL with simpler function approximators learned faster and resulted in more robust policies, as the neural network policies over-fitted on the initial state distribution and did not work well when initialized from different states.

Stochastic gradient descent

While several optimization techniques could be used to fit the parameters of a neural network (e.g. neuroevolution, Koutník et al., 2013), the large number of parameters in most neural networks mean that first-order gradient methods are by far the most popular choice in practice. These techniques calculate an estimate of the first-order gradient of the cost function with respect to all of the network parameters. In the simplest case, the parameters are simply adjusted slightly in the (opposite) direction of the gradient, although often techniques are used that incorporate momentum and adaptive learning rates per parameter such as *rmsprop* (Tieleman and Hinton, 2012) and *adam* (Kingma and Ba, 2014).

Neural networks can learn in a statistically efficient way because their parameters can apply globally and the decomposi-

tion into functions of functions allows the efficient reuse of parameters. While this allows for the generalization of a policy to unexplored parts of the state-space, it also means that the gradient estimates should be representative of the entire state-action space and not biased towards any particular part of it. Therefore, gradient estimates are usually averaged over individual gradients computed for a *batch* of experiences spread out over the state-space. Subsequent gradient estimates should similarly be unbiased; they should be independent and identically distributed (i.i.d.) over the relevant state-action space distribution. When the gradient estimates suffer from high variance (as is the case for Monte-Carlo estimates of the policy gradient, see again Section 3.4), they should be averaged over a larger batch to get a more reliable estimate.

Functions of functions

The assumption that the function that needs to be approximated is composed of a hierarchy of simpler functions is encoded in DNNs by having multiple layers, with each layer computing a function of the outputs of the previous layer. The number of unique functions that the entire network can represent scales exponentially with the number of layers (Raghu et al., 2016) and the optimization of deeper networks has theoretically been shown to be less likely to result in a poor local optimum (Choromanska et al., 2015).

When determining the gradient of the loss function with respect to the parameters, the repeated multiplications with the derivative of a layer with respect to its inputs, resulting from the chain rule, can cause the gradients to become too large or small to effectively learn from. This problem is especially pronounced in recurrent neural networks, which are effectively very deep in time and repeatedly apply the same function (Hochreiter et al., 2001).

Complexity

On domains where the underlying assumptions are valid, DNNs have shown remarkable results in practice. The theoretical foundations are however still somewhat incomplete (Zhang et al., 2016). DRL lacks the theoretical guarantees offered by RL with some other types of function approximators. At the same time, it has been shown to scale to problems where the alternatives are intractable.

The complexity of the interplay of the different components of DRL algorithms makes the learning curve fairly steep for beginning practitioners. Implementation details not mentioned in papers can have a more significant influence on the performance of a method than the parameters that are the focus of the work (Henderson et al., 2017; Tucker et al., 2018). The complexity of the domains DRL is often tested on also contributes to a relatively high computational complexity. This means that DRL papers often include fewer repetitions of the experiments than are needed to get statistically significant results (Henderson et al., 2017).

5.2. Common solution components

A substantial number of DRL algorithms have been proposed recently. These algorithms all have to address the problems outlined in the previous section. To do this, most methods are based on a few shared ideas. While most of these ideas and the problems they address are not limited to RL with DNNs as function approximators, they have proven crucial for getting DRL to work well. This section discusses these common ideas, while the algorithms themselves are discussed in the next section.

Delayed targets

When DRL algorithms use bootstrapping to learn a value function, the learning is posed as a supervised learning problem. For the states and actions in the batch, the targets are the bootstrapped value estimates and the networks are trained by minimizing the difference between the network's predictions and these bootstrapped value estimates. These value targets are problematic for convergence since they are highly correlated with the network predictions. This direct feedback loop can cause the learning process to diverge (Mnih et al., 2015). To ameliorate this problem, the target values can be calculated using an older version of the (action) value function network, often called target network.

Trust region updates

The strongly nonlinear nature of neural networks can mean that a step in parameter space can have an unexpected effect on the behavior of the function. Although small learning rates can help, the resulting increase in training time and required amount of training samples mean that preventing problems in this manner is often infeasible in practice. The problems are especially pronounced for policy gradient strategies based on roll-outs, where the gradients additionally exhibit high variance. Changes to the policy can quickly change the distribution of states visited by the updated policy away from the on-policy distribution for which the update was valid.

To improve the likelihood of the updates to the policy resulting in increased performance, the changes in the policy distribution should therefore be kept small. Several schemes have been proposed to prevent the changes to the parameters of the policy from resulting in too large changes to the policy distribution. These include adding a constraint on the policy distribution change to the optimization (Schulman et al., 2015a), clipping the objective function such that only small changes to the policy distribution are considered beneficial (Schulman et al., 2017), and constraining the policy parameters to be close to the running average of previous policies (Wang et al., 2016).

n-step returns

A problem that is inherent to bootstrapping methods is that they result in biased updates since the targets are based largely on an approximation of a function that should still be learned and is therefore by definition incorrect. This bias can prevent value function based methods from converging. On the other hand, Monte-Carlo based methods, although unbiased, result in high variance. This is because the return calculated for each roll-out trajectory represents only a single sample from the return distribution, while value functions represent the expectation of the return distribution.

On the complex domains that DRL is often applied to, the high variance of Monte-Carlo based methods tends to result in learning that is infeasibly slow. At the same time, the bias of methods based exclusively on learning value functions through bootstrapping results in learning that can be faster at times, while failing to learn anything useful altogether other times. A common strategy therefore is to interpolate between these extremes, for instance by using n-step algorithms (Watkins, 1989). To estimate the return from a certain state, these algorithms use the true rewards observed during n time-steps and the learned value estimate for the state in time step $n+1$. For instance, in n-step SARSA, the action value target becomes:

$$q(x_k, u_k) = r_{k+1} + \gamma r_{k+2} + \cdots + \gamma^{n-1} r_{k+n} + \gamma^n \hat{Q}(x_{k+n+1}, u_{k+n+1}) \tag{31}$$

For $n = 1$, the standard (1-step) SARSA target is recovered, while for $n \to \infty$, (31) becomes a Monte-Carlo estimate of the return. Note that n-step returns are an alternative way to achieve a similar effect to the eligibility traces discussed in Section 2.3. In fact, using eligibility traces leads to a combination of n-step returns for all values of n, exponentially weighted by λ^{n-1} (Sutton and Barto, 2018). The n-step return is preferred in DRL because it tends to be easier to use with momentum based optimization and recurrent neural networks (Mnih et al., 2016).

Just like the use of target networks, the use of n-step return targets reduces the correlations between the value function that is being learned and the optimization targets. Whereas the use of targets networks slows down the learning process in order to attain the convergence gains, the use of n-step returns can speed up learning when the roll-outs are close to on-policy.

Experience replay

One of the largest mismatches between the RL framework and the stochastic gradient descent optimization algorithms used to train DNNs is the requirement of the latter for i.i.d. estimates of the gradients. This requirement can be satisfied by using an experience replay buffer. The consecutive, strongly correlated experiences obtained through interaction with the environment are saved into the buffer. When batches of experiences are needed to estimate the gradients, these batches are assembled by sampling from the buffer in a randomized order, breaking their temporal correlations. The fact that off-policy algorithms can learn about the optimal policy from data obtained by another policy means that a fairly large amount of previous experiences can be retained. This in turn means that even if the policy changes suddenly, the data distribution used to calculate the gradients changes only slowly, which aids with the convergence of the optimization process. Finally, the fact that old experiences can be reused aids the sample efficiency of algorithms using an experience replay buffer. Extensions have also been proposed, with the most popular being to replace uniform sampling from the buffer with sampling based on a distribution determined by the temporal difference error associated with the experiences (Schaul et al., 2016). By sampling surprising experiences more often, the learning process can be sped up significantly. This is similar to the classical idea of prioritized sweeping (Moore and Atkeson, 1993).

When using n-step returns with $n > 1$, it is necessary to compensate for the fact that the samples are not from the policy for which we want to estimate the return. Importance sampling is a popular choice that prevents bias (Precup et al., 2000). The downside of importance sampling is that when the difference between the policies is large, the importance weights quickly become either very small, effectively rendering the sampled experiences useless, or very large, resulting in updates with very high variance. Other compensation strategies that address these issues have been proposed, see Munos et al. (2016) for an overview.

When an on-policy learning algorithm is used, a buffer can be filled with experiences from roll-outs with the policy. After a learning update based on these experiences, the buffer is emptied and the process is repeated.

Input, activation and output normalization

The nonlinearities used in neural networks bound the outputs of the neurons to a certain range. For instance, the popular Rectified Linear Unit (ReLU) maps all non-positive inputs to zero. As a consequence, when calculating the derivatives of these nonlinearities with respect to their inputs, this derivative can be very small when the input is outside of a certain range. For the ReLU, the derivative of the activation with respect to all parameters that led to the activation is zero when the input to the ReLU was non-positive. As a consequence, none of the parameters that led to the activation will be updated, regardless of how wrong the activation was. It is therefore important that the inputs to all neural network layers (whether they be the input to the network or the ouputs of previous layers) are within a sensible range.

When the properties of the inputs are unknown a priori and they cannot be normalized manually, adaptive normalization can be used. These techniques can also be used on subsequent layers. Normalization techniques include batch normalization (Ioffe and Szegedy, 2015), layer normalization (Ba et al., 2016) and weight normalization (Salimans and Kingma, 2016).

Similar considerations apply to the backward pass through a network during training. The gradients of the loss with respect to the parameters should not be too large, as an update based on large gradients can quickly cause the subsequent activations of the unit with the updated parameters to be outside of this range for which learning works well. Particularly, this means that while the scale of the reward function does not influence most forms of RL, DRL algorithms can be sensitive to this property (Henderson et al., 2017).

To ensure that the parameter gradients are within a sensible range, these gradients are often clipped. This changes the optimization objective but prevents destructive updates. Additionally, when learning value functions, the reward function can be scaled such that the resulting value function is of a sensible order of magnitude. Rewards are also sometimes clipped, although this changes the problem definition. Finally, the target values can be adaptively normalized during learning (van Hasselt et al., 2016a).

Table 1: Deep reinforcement learning algorithms reviewed. Return estimation refers to the targets for value functions and / or the return estimation in the policy gradient. Update constraints refer to both constraints on bootstrapping as well ass updates to the policy.

Algorithm	Policy	Return estimation	Update constraints	Data distribution
NFQ	discrete[3], deterministic	1-step Q	bootstrap with old θ	off-policy fixed apriori
(D)DQN	discrete, deterministic	1-step Q	bootstrap with old θ	off-policy experience replay
DDPG	continuous, deterministic	1-step Q	bootstrap with old θ, w	off-policy experience replay
TRPO	discrete / continuous stochastic	∞-step Q	policy constraint	on-policy
PPO	discrete / continuous stochastic	n-step advantage (GAE)	clipped objective	on-policy
A3C	discrete / continuous stochastic	n-step advantage	-	on-policy
ACER	discrete / continuous stochastic	n-step advantage	average policy network	on-policy + off-policy

5.3. Popular DRL algorithms

In this section we will discuss some of the more popular or historically relevant algorithms for deep reinforcement learning. These algorithms all address the challenges of performing RL with (deep) neural network function approximation by combining implementations of some of the ideas outlined in the previous section. Table 1 gives a comparison of some popular or historically relevant DRL algorithms.

Neural Fitted Q iteration (NFQ)

An important early development in achieving convergent RL with neural network function approximation was the Neural Fitted Q iteration (NFQ) algorithm (Riedmiller, 2005), a variant of fitted-Q iteration (Algorithm 6). The algorithm uses a fixed experience buffer of previously obtained interaction samples from which to sample randomly. By calculating the target Q-values for all states at the start of each optimization iteration, the optimization is further helped to converge. A final measure to aid convergence was to add artificial experience samples to the database at the goal states, where the true Q-values were known.

Deep Q-network (DQN)

While good for convergence, the need for an a-priori fixed set of experiences is limiting. While new experiences can be added to the NFQ buffer, Mnih et al. (2015) proposed to continuously write experiences to an experience replay buffer during training, and to sample experiences uniformly at random from this buffer at regular environment interaction intervals. Since the constant changes to the contents of the buffer and the learned Q-function mean that good targets can not be calculated a priori, a copy θ^- of the Q-function parameters θ is kept in memory. The optimization targets (Q-values) are calculated using a target network, which is a copy of the Q-function network using these older parameters θ^-. At regular intervals the target network parameters θ^- are updated to be equal to the current parameters θ. Mnih et al. (2015) demonstrated their method using raw images as inputs. Their convolutional Deep Q-Network (DQN) achieved super-human performance on a number of Atari games, resulting in growing interest in the field of DRL.

[3] A different version of NFC for continuous actions (NFQ-CA) does exist (Hafner and Riedmiller, 2011).

The base DQN algorithm is simple to implement. Through various extensions, DQN can achieve competitive performance on domains with discrete actions (Hessel et al., 2017).

Double DQN (DDQN)

Although value function based methods are inherently biased, DQN suffers from a particular source of bias that can be reduced fairly easily. This form of bias is the overestimation of the returns which results from the maximization over the Q-values (7). The max operator uses the same values to both select and evaluate the Q-values, which makes over-estimation of the values likely (van Hasselt et al., 2016b). To address this problem, the selection and evaluation can be decoupled. The original double Q-learning algorithm did this by learning two separate Q-functions, based on separate experiences (van Hasselt, 2010). One of these Q-functions is then used for the action selection while the other is used to determine the Q-value for that action. The Double Deep Q Network (DDQN) algorithm (van Hasselt et al., 2016b) uses the two separate networks that are already used in DQN for the separation such that the complexity of the algorithm is not increased. As in DQN, the target network is used to determine the value of the Q-function used for bootstrapping, while the on-line network is used to determine for which action the target Q-function is evaluated. This makes the optimization targets:

$$q(x, u) = r + \gamma \hat{Q}\left(x', \arg\max_{u'} \hat{Q}(x', u'; \theta); \theta^-\right).$$

This simple change was shown to improve the convergence and performance of the DQN algorithm.

Deep Deterministic Policy Gradient (DDPG)

For continuous action spaces, an actor-critic algorithm exists that is closely related to DQN. This Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2015) uses a deterministic policy $u = \hat{\pi}(x; w)$. For convergence, target network copies of both the actor and the critic are used for the critic's optimization targets:

$$q(x, u) = r + \gamma \hat{Q}\left(x', \hat{\pi}(x'; w^-); \theta^-\right).$$

In this algorithm the target network parameters θ^-, w^- slowly track the online parameters θ, w using a low pass filter. They are updated after each optimization step according to:

$$\theta^- \leftarrow (1 - \tau)\theta^- + \tau\theta$$
$$w^- \leftarrow (1 - \tau)w^- + \tau w,$$

with $\tau \ll 1$. To calculate the gradients for updating the policy parameters, the algorithm uses samples of the deterministic policy gradient (Silver et al., 2014):

$$\nabla_w J \approx \frac{1}{B} \sum_b \nabla_a \hat{Q}(x, u; \theta)|_{x=x_b, u=\hat{\pi}(x_b;w)} \nabla_w \hat{\pi}(x; w)|_{x=x_b}, \quad (32)$$

with b the index in the mini-batch of size B containing experiences sampled uniformly at random from the experience buffer $\mathcal{S}$. The DDPG method additionally uses batch normalization layers (Ioffe and Szegedy, 2015).

DDPG is one of the simpler DRL algorithms allowing for continuous action spaces. Since the algorithm is off-policy, it additionally allows for experience replay, which together with the use of bootstrapping can lead to sample efficient learning. However, its off-policy nature makes DDPG most suitable for domains with stable dynamics (Henderson et al., 2017). Additionally, the bias in the policy gradient due to the exclusive reliance on a learned value function can limit the performance and convergence of the algorithm.

Trust Region Policy Optimization (TRPO)

While DDPG uses an off-policy critic to determine the policy gradient for a deterministic policy, Schulman et al. (2015a) introduced a policy gradient method on the other end of the bias-variance spectrum. Their Trust Region Policy Optimization (TRPO) algorithm uses a large number of roll-outs with the current policy to obtain state action pairs with Monte Carlo estimates of their returns $\hat{Q}^{w_{old}}(x, u)$. The stochastic policy is then updated by optimizing for the conservative policy optimization objective (Kakade and Langford, 2002), while constraining the difference between the policy distribution after the optimization and the older policy distribution used to obtain the samples:

$$\max_w E\left\{\frac{\hat{\tilde{\pi}}(x, u; w)}{\hat{\tilde{\pi}}(x, u; w_{old})} \hat{Q}^{w_{old}}(x, u)\right\} \quad (33)$$

$$\text{subject to } E\left\{D_{KL}(\hat{\tilde{\pi}}(x, \cdot; w_{old}) \parallel \hat{\tilde{\pi}}(x, \cdot; w))\right\} \leq c \quad (34)$$

where D_{KL} denotes the Kullback-Leibler divergence, and the expectations are with respect to the state distribution induced by the old policy. To perform the optimization, a linear approximation is made to the objective and a quadratic approximation is made to the constraint. The conjugate gradient method is then used followed by a line search to calculate the next parameter values. The TRPO method is relatively complicated and sample inefficient, but does provide relatively reliable improvements to the policy.

Generalized Advantage Estimation (GAE)

The stochastic policy gradient can be written as (Schulman et al., 2015b):

$$\nabla_w J = E\{\sum_{k=0}^{\infty} \Psi_k \nabla_\theta \log \tilde{\pi}(x_k, u_k; w)\}, \quad (35)$$

where Ψ_k is an estimate of the return when taking action u_k in state x_k and following the policy afterwards. A trade-off between the bias and variance of the policy gradient estimates can be made by choosing how much Ψ_k is based on observed rewards versus a learned value estimate, as discussed in Section 5.2. Additionally, the variance of the policy gradient can be reduced by subtracting a baseline from the return estimate (Greensmith et al., 2004). A common and close to optimal choice for the baseline is the state-value function. This makes Ψ the advantage function:

$$A(x, u) = Q(x, u) - V(x),$$

which represents the advantage of taking action u in state x as opposed to the policy action $\pi(x)$. An n-step estimate of the advantage function is:

$$\hat{A}^{(n)}(x_k, u_k) = \sum_{i=1}^{n} \gamma^{i-1} r_{k+i} + \gamma^n \hat{V}(x_{k+n+1}; \theta) - \hat{V}(x_k; \theta) \quad (36)$$

To trade off the bias introduced by the imperfect learned value function for low n with the variance of estimators with high n, Schulman et al. (2015b) define a Generalized Advantage Estimator (GAE) as an exponentially weighted average of n-step advantage estimators:

$$\hat{A}^{GAE(\lambda)} := (1 - \lambda) \sum_{n=1}^{N} (\lambda^{n-1} \hat{A}^{(n)}) \quad (37)$$

The authors use the estimator with the TRPO algorithm. The value function is learned from Monte-Carlo estimations with trust region updates as well.

Proximal Policy Optimization (PPO)

The constrained optimization of TRPO makes the algorithm relatively complicated and prevents using certain neural network architectures. In the Proximal Policy Optimization (PPO) algorithm, Schulman et al. (2017) therefore replace the hard constraint by a clipped version of the objective function, which ensures that for each state the potential gain from changing the state distribution is limited, while the potential loss is not. This allows optimizing the objective (which uses the GAE) with SGD-based techniques, as well as adding additional terms to the objective. Specifically, a regression loss for the value function is added, which allows parameter sharing between the value function and policy. Additionally, a loss based on the entropy of the policy is added to encourage exploration (Williams and Peng, 1991). PPO is a relatively simple algorithm that offers competitive performance.

Asynchronous Advantage Actor Critic (A3C)

Instead of collecting a large number of consecutive on-policy trajectories with a single policy, which are then batched together, Mnih et al. (2016) proposed the use of a number of parallel actors with global shared parameters. These actors all calculate updates with respect to the shared parameters, which they apply to the parameters asynchronously (Recht

et al., 2011). To ensure the actors explore different parts of the state-action space so that the parameter updates better meet the i.i.d. assumption, each agent uses a different exploration policy. While a number of proposed algorithms benefited from the parallel actor setup, the most successful was the Asynchronous Advantage Actor Critic (A3C) algorithm. This algorithm takes a small number of steps, after which it calculates n-step advantage estimates (37) and value function estimates for these roll-out steps. These are then used to calculate gradients to update the policy (35) and the value function.

Actor Critic with Experience Replay (ACER)

The downside of the on-policy methods (TRPO, PPO, A3C) is that once a step has been made in policy space, reevaluating the policy gradient requires discarding all previous experiences and running trials with the new policy. To increase the sample efficiency, it is desirable to combine the good convergence of the on-policy algorithms with the ability to reuse past experiences of off-policy algorithms.

One algorithm that does this is the Actor Critic with Experience Replay (ACER) algorithm of Wang et al. (2016). It uses the A3C algorithm as a base and combines it with a trust region update scheme based on limiting the distance between the new policy parameters and those of a running average of recent policies. It then alternates between the standard on-policy updates of A3C and off-policy updates, where each parallel agent samples trajectories from a local experience buffer for the updates. Truncated importance sampling with a bias correction term is used to correct for the off-policy nature of the n-step trajectories. While the algorithm offers very competitive performance for both discrete and continuous actions, it is relatively complex.

Interpolated Policy Gradient (IPG)

Another way in which on- and off-policy algorithms can be combined is to simply interpolate between the biased yet sample efficient deterministic policy gradient obtained from an off-policy critic (32) and the unbiased yet sample inefficient on-policy Monte Carlo estimate of the policy gradient. This Interpolated Policy Gradient (IPG) method was proposed by Gu et al. (2017), who found intermediate (but mostly on-policy) ratios to work best.

5.4. Extensions

The DRL algorithms discussed in the previous section mostly address the pitfalls of combining RL with DNNs. However, the use of DNNs also offers opportunities to go beyond simply performing RL with DNN function approximation. The functional decomposition of DNNs means that while later layers might compute very task specific features, earlier layers could represent much more general functions. For example, while later layers in a convolutional network might learn to recognize task specific objects, earlier layers might learn to detect edges or textures (Olah et al., 2017). These earlier layers might therefore easily generalize to new tasks and, equivalently, be trained from data obtained from separate tasks. Therefore, the deep learning assumptions make the combination of DRL with transfer learning and state representation learning very interesting.

State representation learning

In most of this survey, we have considered the standard RL problem in which the agent has access to the state of the environment x. In real control applications, especially in the domain of robotics, the state of the environment is not directly accessible. Instead, only some indirect effects of the true environment state might be observed by a set of sensors. Without resorting to the POMDP formalism (discussed later), learning a policy in this case can therefore be seen as a combination of learning a representation of the state from the sensor data and learning a policy based on the state representation. While the state representation can be learned implicitly through DRL, the number of required trial and error samples might be prohibitively expensive as the reward signal might contain only very indirect information on how to learn the state-representation.

Instead, explicit State Representation Learning (SL) objectives can be used before or during the RL phase. These objectives can allow learning from unlabeled sensor data, as well as limiting the parameter search space through the inclusion of prior knowledge. Auto-encoding is a popular SRL objective as it is fully unsupervised; through a compression objective salient details are extracted from observations that are highly redundant (Hinton and Salakhutdinov, 2006; Lange et al., 2012; Finn et al., 2016). Besides the knowledge that observations are highly redundant, other priors include the fact that the state of the world only changes slowly over time (Wiskott and Sejnowski, 2002), as well as the fact that the state should be predictive of immediate received rewards (Shelhamer et al., 2016). Additional priors, relevant to physical domains, were suggested by Jonschkowski and Brock (2015). Besides encoding general knowledge about the state of the world, it is possible to learn to encode the observations in a way that is suitable for control. One example is the work of Watter et al. (2015), which embeds images into a state-space in which actions have a (locally) linear effect. Another example is provided by Jonschkowski et al. (2017) who learned to encode the positions and velocities of relevant objects in an unsupervised manner. Jaderberg et al. (2017) proposed to learn, off-policy, additional value functions for optimizing pseudo rewards based on controlling the sensory observations and the activations of the neurons of the networks. The inclusion of SL in DRL can help learn representations and policies that generalize to unseen parts of the state-space more easily (de Bruin et al., 2018).

Transfer learning

Just as the generality of the functions encoded by the earlier layers of the policy and value function DNNs means that they can be trained with more than just RL updates, and generalize to unseen parts of the state-space, it also means that the encoded functions can be relevant to more than just the training task. This makes DRL suitable for transfer learning, where generalization needs to be performed to a new task, rather than just across the state-space of the training task. In this context, Parisotto et al. (2015) used DQN agents trained on several Atari

games as teachers for a separate DQN agent that was trained to output similar actions, and have similar internal activations as the teachers. This was found to result in a weight initialization that sped up learning on new games significantly, given enough similarity between the new games and some of the training games. It is also possible to more explicitly parameterize representations for transfer. Universal Value Functions (UVFs) (Schaul et al., 2015) are one example where value functions are learned that generalize over both states and goal specifications. To improve the performance in domains where only reaching a goal results in obtaining a reward, Andrychowicz et al. (2017) proposed Hindsight Experience Replay (HER), which relabels a failed attempt to reach a certain goal as a successful attempt to reach another goal. Another representation that is suitable for transfer learning is the Successor Features (SF) representation (Barreto et al., 2017) which is based on successor representations (Dayan, 1993). These representations decouple the value function into a representation of the discounted state distribution induced by the policy and the rewards obtained in those states. Zhang et al. (2017) showed the use of this representation with DRL in the robotics domain.

Supervised policy representation learning

Sometimes the state of the environment is available for specific training cases, but not in general. For instance, a robot might be placed in a motion capture arena. In this case, it might be relatively simple to learn or calculate the correct actions for the states in the arena. Alternatively, it might be possible to solve the RL problem from specific initial states, but hard to learn a general policy for all initial states. In both of these scenarios, trajectories of observations and actions can be collected and supervised learning can be used to train DNN policies that generalize to the larger state-space, preventing many of the issues of DRL. One technique that applies this principle is Guided Policy Search (GPS), which adds a constraint to the local controllers on the deviation from the global policy, such that the local policies do not give solutions that the DNN can not learn to represent (Levine and Koltun, 2013; Levine et al., 2016).

6. Outlook

We close our review with an outlook that starts by touching on important areas of (or related to) RL that we could not cover in our main survey. Then, we explain some ways in which practical problems may violate the standard MDP formulation of the problem, and – where available – point out generalized methods that address this. Finally, we signal some important issues that remain open for RL methods.

6.1. Research areas

There are entire fields of research that contribute ideas or algorithms to RL, but that we were unable to cover in this review. These fields include among others robotics (Deisenroth et al., 2011; Kober et al., 2013), operations research (Powell, 2012), economics (Kamien and Schwartz, 2012), and neuroscience (Sutton and Barto, 2018, Ch. 15). Within control, relevant subfields include optimal control, adaptive control, and model-predictive control, which we touched on briefly in Section 4; in addition to other more specific areas like iterative learning control (Moore, 2012) or extremum seeking (Ariyur and Krstic, 2003). A specific area of AI research with deep connections to RL and receding-horizon MPC is online or sample-based planning, which at each step uses a model to simulate and evaluate several candidate sequences of actions, or closed-loop action selection rules in the stochastic case (Kocsis and Szepesvári, 2006; Weinstein and Littman, 2012; Buşoniu et al., 2012; Munos, 2014). Then, one of these solutions is selected, its first step is applied, and the procedure is repeated in the next state. These methods trade off a curse of dimensionality with respect to the state and action size, with a "curse of horizon" – they are generally exponentially complex in the horizon up to which sequences are examined (Munos, 2014).

A crucial component of RL that we discussed only briefly is exploration. Exploration methods can be grouped into undirected and directed exploration (Thrun, 1992). While undirected methods indiscriminately apply some noise to the action selection, with the prototypical example being ε-greedy exploration (12), directed exploration methods use knowledge of the learning process to explore in a smarter manner. For example, methods like Bayesian RL (Ghavamzadeh et al., 2015; Russell and Norvig, 2016) and bandit theory (Auer et al., 2002) offer principled ways of designing and analyzing exploration strategies. Another benefit of Bayesian RL is the easier incorporation of prior knowledge into the algorithms. Other directed exploration methods include those that add to the original rewards an extra exploration-inducing term, called intrinsic reward, when visiting states that are deemed interesting (Barto, 2013). These intrinsic rewards can for example be based on the (pseudo) state visit count (Bellemare et al., 2016a), on the temporal difference error (Achiam and Sastry, 2017) or on the prediction accuracy of a simultaneously learned dynamics model (Schmidhuber, 1991). Note that these methods imply additional computational costs, which may be significant for some of them, like Bayesian RL. In ADP approaches, exploration is often called probing noise.

6.2. Generalizing the problem

The underlying models used by most of the RL algorithms discussed above assume noise-free state information, whilst many control processes possess output feedback buried in noise and prone to delays (Jaakkola et al., 1995; Bertsekas, 2017; Azizzadenesheli et al., 2016; Tolić et al., 2012; Bai et al., 2012). This imperfect state information can be soundly handled within the framework or Partially Observable Markov Decision Processes (POMDPs) (Jaakkola et al., 1995; Azizzadenesheli et al., 2016; Bai et al., 2012; Russell and Norvig, 2016) or – when at least some model information is available – by using state estimation (Bertsekas, 2017; Tolić et al., 2012; Tolić and Palunko, 2017). However, solving POMDPs imposes significantly greater computational costs than MDP-based RL (Bai

et al., 2012). Like in RL, the exploitation-exploration issue can be addressed using e.g. Bayesian RL ideas (Ross et al., 2011).

All models utilized in this paper are time-invariant (stationary using the AI vocabulary). Therefore, provided that the underlying model changes "slowly enough", all algorithms presented herein readily apply. However, the precise characterization of "slowly enough" intricately depends on the algorithm learning rate and additional results are needed in this regard. Despite our efforts, we were not able to find any work focusing exclusively on this topic. General remarks and guidelines are found in (Powell, 2012; Russell and Norvig, 2016; Bertsekas, 2017; Sutton and Barto, 2018). It is also of interest to investigate models with delayed dynamics, which also appears to be an uninvestigated problem in RL.

Another challenge with the standard MDP formulation arises when the sampling frequency of the system to be controlled is high. Higher frequencies mean that the effect of a single action on the eventual return reduces. Using the difference in expected returns for different actions to determine a policy therefore becomes problematic, especially when combined with function approximation or when noise is present. From a control perspective however, a sufficiently high sampling frequency can be crucial for the performance of the controller and for disturbance rejection (Franklin et al., 1998). While RL works often consider the sampling frequency to be a given property of the problem, in reality it is an important meta-parameter that needs to be selected. While more work is needed in this direction, there are approaches that make RL more suitable for higher sampling frequencies, or even continuous time. These include the Semi-Markov Decision Process (SMDP) framework (Bradtke and Duff, 1995), which adds the time it takes to transition between states to the MDP framework, as well as the advantage learning algorithm (Baird, 1999) and the consistent Bellman operator (Bellemare et al., 2016b) which both devalue suboptimal actions in order to increase the difference between the expected returns of the optimal and sub-optimal actions (the action gap).

Multi-agent decentralized RL, in which the agents learn to optimize a common performance index, is not yet fully solved (Buşoniu et al., 2008; Lewis et al., 2012; Beuchat et al., 2016; Russell and Norvig, 2016; Tolić and Palunko, 2017). When it comes to adversarial agents, the game theoretic viewpoint is needed (Lewis et al., 2012; Modares et al., 2015; Russell and Norvig, 2016). From the stability and learning convergence point of view, impediments of multi-agent games range from the existence of multiple equilibria to non-stationary costs-to-go owing to coupled problems.

6.3. Other open issues

An open issue that applies to AI and control equally is the design of the right function approximator for a specific problem. Simpler architectures, like basis functions, suffer more from this problem since they are less flexible; while more general architectures like (deep) neural networks, kernel representations, Gaussian processes, support vector regression, regression trees, etc. have just a few meta-parameters to tune and are (at least ideally) less sensitive to tuning. Nevertheless, using these more complicated representations may not always be an option, due e.g. to computational restrictions, or even fundamental ones – many ADP methods with stability analysis only work for linear BF expansions, for instance. So the question of choosing the right BFs is still a relevant one, see e.g. Munos and Moore (2002); Grüne (2004); Bertsekas and Yu (2009).

A related issue is that, in general, RL can still only handle small-to-medium scale problems, up to on the order of ten variables. This limit is broken in certain cases where specific assumptions may be made on the structure of the state signal – e.g. deep RL can handle image (or image-like) state signals with tens or hundreds of thousands of pixels. The scale drops severely when more complicated flavors of problems, like POMDPs, have to be solved. An alternative pathway to scalability may be provided by hierarchical RL (Barto and Mahadevan, 2003).

Control-related deficiencies of RL, some of which are discussed by Khargonekar and Dahleh (2018), include lack of transparency and interpretability as well as vulnerability to adversarial attacks and to rapid and unforeseen changes in the environments. Regarding interpretability for instance, once the learning process of a model-free algorithm is over, the Q-function encodes (i.e., hides away) the underlying model. In other words, neither the structure of the model (e.g., number of the states, dominant dynamics/eigenvalues, etc.) nor its parameters are discovered. Moreover, it is not clear how to detect when a model parameter changes slightly in such a way that the underlying model becomes unstable. A novel learning algorithm with novel assumptions (e.g., novel admissible policies) might be required. In addition, it is not clear how to determine (and address) the case when exploration during the learning process was not adequate and some important dynamics were not learned.

In deep RL, recent public successes have led to a substantial increase of interest in the field over the last few years. While DRL is certainly very capable and can give very good results when applied to the right problems, it is not the right tool for every problem. In the context of control, more research is needed into determining which kinds of problems benefit from using DNNs as function approximators. This might have to be done by getting a better understanding of how the representations are learned for common network types and what types of functions are represented and learned efficiently. A better understanding of the representation of a learned policy might also help with the interpretability issue, which is even more problematic for deep representations than for classical ones, and can therefore lead to a major bottleneck for using DRL in control. Additional gains in the convergence properties and analysis of DRL algorithms are also needed, as DRL methods are often quite sensitive to their meta-parameters. Furthermore, the resulting controllers lack any types of stability guarantees and have been shown to over-fit to the training policy trajectories. Finally, continued work on combining on and off-policy approaches will hopefully lead to algorithms that yield improved stability with better sample efficiency.

Despite the fact that ADP approaches consider stability, many challenges remain, so that general stable RL is unsolved

(Section 4). To address this, stability analysis should identify the most general possible assumptions on the dynamics, and requirements on the rewards, so that (near-)optimal solutions are stabilizing. Within these (hopefully, not very tight) constraints imposed by stability, AI should then take over and provide algorithms that converge quickly to near-optimal solutions. These algorithms should not be fully black-box, but should exploit any model and stability knowledge provided by control theory. How to do this is not yet known, especially when the dynamics or exploration are stochastic, but we strongly believe that this is a very promising area, and that good solutions can only arise from a synergy of control-theoretic stability and AI optimality.

Acknowledgments

The work of L. Buşoniu is supported by a grant of the Romanian National Authority for Scientific Research, CNCS-UEFISCDI, project number PN-III-P1-1.1-TE-2016-0670, grant agreement no. 9/2018. The work of D. Tolić and I. Palunko is supported by Croatian Science Foundation under the project IP-2016-06-2468 "ConDyS". This work is additionally part of the research programme Deep Learning for Robust Robot Control (DL-Force) with project number 656.000.003, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

References

Achiam, J., Sastry, S., 2017. Surprise-based intrinsic motivation for deep reinforcement learning. arXiv preprint arXiv:1703.01732.

Amari, S., Douglas, S. C., 1998. Why Natural Gradient? In: Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing. Seattle, USA, pp. 1213–1216.

Anderson, C. W., Young, P. M., Buehner, M. R., Knight, J. N., Bush, K. A., Hittle, D. C., July 2007. Robust reinforcement learning control using integral quadratic constraints for recurrent neural networks. IEEE Transactions on Neural Networks 18 (4), 993–1002.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., Zaremba, W., 2017. Hindsight experience replay. In: Advances in Neural Information Processing Systems. pp. 5048–5058.

Ariyur, K. B., Krstic, M., 2003. Real-time optimization by extremum-seeking control. John Wiley & Sons.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., Bharath, A. A., 2017. A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866.

Auer, P., Cesa-Bianchi, N., Fischer, P., 2002. Finite-time analysis of the multiarmed bandit problem. Machine Learning 47 (2-3), 235–256.

Azizzadenesheli, K., Lazaric, A., Anandkumar, A., 2016. Reinforcement learning of POMDPs using spectral methods. In: COLT. Vol. 49 of JMLR Workshop and Conference Proceedings. JMLR.org, pp. 193–256.

Ba, J. L., Kiros, J. R., Hinton, G. E., 2016. Layer normalization. arXiv preprint arXiv:1607.06450.

Bai, H., Hsu, D., Kochenderfer, M. J., Lee, W. S., 2012. Unmanned aircraft collision avoidance using continuous-state pomdps. Robotics: Science and Systems VII 1, 1–8.

Baird, L. C., 1999. Reinforcement learning through gradient descent. Ph.D. thesis, Carnegie Mellon University.

Barreto, A., Munos, R., Schaul, T., Silver, D., 2017. Successor features for transfer in reinforcement learning. Neural Information Processing Systems (NIPS).

Barto, A., Mahadevan, S., 2003. Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems: Theory and Applications 13 (4), 341–379.

Barto, A. G., 2013. Intrinsic motivation and reinforcement learning. In: Intrinsically motivated learning in natural and artificial systems. Springer, pp. 17–47.

Barto, A. G., Sutton, R. S., Anderson, C. W., 1983. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. IEEE Transactions on Systems, Man, and Cybernetics 13 (5), 834–846.

Baxter, J., Bartlett, P. L., 2001. Infinite-horizon policy-gradient estimation. Journal of Artificial Intelligence Research 15, 319–350.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R., 2016a. Unifying count-based exploration and intrinsic motivation. In: Advances in Neural Information Processing Systems (NIPS). pp. 1471–1479.

Bellemare, M. G., Ostrovski, G., Guez, A., Thomas, P. S., Munos, R., 2016b. Increasing the action gap: New operators for reinforcement learning. In: AAAI. pp. 1476–1483.

Bengio, Y., Delalleau, O., Roux, N. L., 2006. The curse of highly variable functions for local kernel machines. In: Advances in neural information processing systems. pp. 107–114.

Bertsekas, D. P., 2005. Dynamic programming and suboptimal control: A survey from ADP to MPC. European Journal of Control 11 (4), 310 – 334.

Bertsekas, D. P., 2012. Dynamic Programming and Optimal Control, 4th Edition. Vol. 2. Athena Scientific.

Bertsekas, D. P., 2017. Dynamic Programming and Optimal Control, 4th Edition. Vol. 1. Athena Scientific.

Bertsekas, D. P., Shreve, S. E., 1978. Stochastic Optimal Control: The Discrete Time Case. Academic Press.

Bertsekas, D. P., Tsitsiklis, J. N., 1996. Neuro-Dynamic Programming. Athena Scientific.

Bertsekas, D. P., Yu, H., 30 March – 2 April 2009. Basis function adaptation methods for cost approximation in MDP. In: Proceedings 2009 IEEE Symposium on Approximate Dynamic Programming and Reinforcement Learning (ADPRL-09). Nashville, US, pp. 74–81.

Beuchat, P., Georghiou, A., Lygeros, J., 2016. Approximate dynamic programming: a Q-function approach. http://arxiv.org/abs/1602.07273.

Bhatnagar, S., Sutton, R., Ghavamzadeh, M., Lee, M., 2009. Natural actor-critic algorithms. Automatica 45 (11), 2471–2482.

Borrelli, F., Bemporad, A., Morari, M., 2017. Predictive Control for Linear and Hybrid Systems. Cambridge University Press, Cambridge, UK.

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

Bradtke, S. J., Duff, M. O., 1995. Reinforcement learning methods for continuous-time markov decision problems. In: Advances in neural information processing systems (NIPS). pp. 393–400.

Buşoniu, L., Babuška, R., De Schutter, B., 2008. A comprehensive survey of multi-agent reinforcement learning. IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews 38 (2), 156–172.

Buşoniu, L., Babuška, R., De Schutter, B., Ernst, D., 2010. Reinforcement Learning and Dynamic Programming Using Function Approximators. Automation and Control Engineering. Taylor & Francis CRC Press.

Buşoniu, L., Lazaric, A., Ghavamzadeh, M., Munos, R., Babuška, R., De Schutter, B., 2011. Least-squares methods for policy iteration. In: Wiering, M., van Otterlo, M. (Eds.), Reinforcement Learning: State of the Art. Vol. 12 of Adaptation, Learning, and Optimization. Springer, pp. 75–109.

Buşoniu, L., Munos, R., Babuška, R., 2012. A review of optimistic planning in Markov decision processes. In: Lewis, F., Liu, D. (Eds.), Reinforcement Learning and Adaptive Dynamic Programming for Feedback Control. Wiley.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., LeCun, Y., 2015. The loss surfaces of multilayer networks. In: Artificial Intelligence and Statistics. pp. 192–204.

Dayan, P., 1993. Improving generalization for temporal difference learning: The successor representation. Neural Computation 5 (4), 613–624.

de Bruin, T., Kober, J., Tuyls, K., Babuška, R., 2018. Integrating state representation learning into deep reinforcement learning. IEEE Robotics and Automation Letters 3 (3), 1394–1401.

de Farias, D. P., Roy, B. V., 2003. The linear programming approach to approximate dynamic programming. Operations Research 51 (6), 850–865.

Deisenroth, M., Neumann, G., Peters, J., 2011. A survey on policy search for robotics. Foundations and Trends in Robotics 2 (1–2), 1–141.

Diehl, M., Amrit, R., Rawlings, J. B., 2011. A lyapunov function for economic optimizing model predictive control. IEEE Transactions on Automatic Control 56 (3), 703–707.

Engel, Y., Mannor, S., Meir, R., 7–11 August 2005. Reinforcement learning

with Gaussian processes. In: Proceedings 22nd International Conference on Machine Learning (ICML-05). Bonn, Germany, pp. 201–208.

Ernst, D., Geurts, P., Wehenkel, L., 2005. Tree-based batch mode reinforcement learning. Journal of Machine Learning Research 6, 503–556.

Ernst, D., Glavić, M., Capitanescu, F., Wehenkel, L., 2009. 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.

Farahmand, A.-m., Ghavamzadeh, M., Szepesvári, C., Mannor, S., 2016. Regularized policy iteration with nonparametric function spaces. The Journal of Machine Learning Research 17 (1), 4809–4874.

Farahmand, A. M., Ghavamzadeh, M., Szepesvári, Cs., Mannor, S., 2009. Regularized policy iteration. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (Eds.), Advances in Neural Information Processing Systems 21. MIT Press, pp. 441–448.

Finn, C., Tan, X. Y., Duan, Y., Darrell, T., Levine, S., Abbeel, P., 2016. Deep spatial autoencoders for visuomotor learning. In: Robotics and Automation (ICRA), 2016 IEEE International Conference on. IEEE, pp. 512–519.

Franklin, G. F., Powell, D. J., Workman, M. L., 1998. Digital Control of Dynamic Systems. Vol. 3. Addison-wesley Menlo Park.

Friedrich, S. R., Buss, M., 2017. A robust stability approach to robot reinforcement learning based on a parameterization of stabilizing controllers. In: International Conference on Robotics and Automation (ICRA). pp. 3365–3372.

Geist, M., Pietquin, O., 2013. Algorithmic survey of parametric value function approximation. IEEE Transactions on Neural Networks and Learning Systems 24 (6), 845–867.

Ghavamzadeh, M., Mannor, S., Pineau, J., Tamar, A., 2015. Bayesian reinforcement learning: A survey. Foundations and Trends in Machine Learning 8 (5-6), 359–492.

Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y., 2016. Deep learning. Vol. 1. MIT press Cambridge.

Gordon, G., 9–12 July 1995. Stable function approximation in dynamic programming. In: Proceedings 12th International Conference on Machine Learning (ICML-95). Tahoe City, US, pp. 261–268.

Görges, D., 2017. Relations between model predictive control and reinforcement learning. IFAC-PapersOnLine 50 (1), 4920 – 4928, 20th IFAC World Congress.

Gosavi, A., 2009. Reinforcement learning: A tutorial survey and recent advances. INFORMS Journal on Computing 21 (2), 178–192.

Greensmith, E., Bartlett, P. L., Baxter, J., 2004. Variance reduction techniques for gradient estimates in reinforcement learning. Journal of Machine Learning Research 5 (Nov), 1471–1530.

Grimm, G., Messina, M., Tuna, S., Teel, A., 2005. Model predictive control: For want of a local control Lyapunov function, all is not lost. IEEE Transactions on Automatic Control 50 (5), 546–558.

Grondman, I., Buşoniu, L., Lopes, G., Babuška, R., 2012. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics 42 (6), 1291–1307.

Grüne, L., 2004. Error estimation and adaptive discretization for the discrete stochastic Hamilton-Jacobi-Bellman equation. Numerische Mathematik 99 (1), 85–112.

Grüne, L., Pannek, J., 2016. Nonlinear Model Predictive Control: Theory and Algorithms, 2nd Edition. Springer.

Gu, S., Lillicrap, T., Turner, R. E., Ghahramani, Z., Schölkopf, B., Levine, S., 2017. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 3849–3858.

Hafner, R., Riedmiller, M., 2011. Reinforcement learning in feedback control. Machine learning 84 (1-2), 137–169.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D., 2017. Deep reinforcement learning that matters. arXiv preprint arXiv:1709.06560.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., Silver, D., 2017. Rainbow: Combining improvements in deep reinforcement learning. arXiv preprint arXiv:1710.02298.

Hinton, G. E., Salakhutdinov, R. R., 2006. Reducing the dimensionality of data with neural networks. Science 313 (5786), 504–507.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al., 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.

Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural networks 4 (2), 251–257.

Ioffe, S., Szegedy, C., 2015. Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Int. Conf. Machine Learning (ICML).

Jaakkola, T., Jordan, M. I., Singh, S. P., 1994. On the convergence of stochastic iterative dynamic programming algorithms. Neural Computation 6 (6), 1185–1201.

Jaakkola, T., Singh, S. P., Jordan, M. I., 1995. Reinforcement learning algorithm for partially observable markov decision problems. In: Advances in Neural Information Processing Systems 7. MIT Press, pp. 345–352.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., Kavukcuoglu, K., 2017. Reinforcement learning with unsupervised auxiliary tasks. In: Int. Conf. Learning Representations (ICLR).

Jiang, Y., Jiang, Z.-P., 2012. Computational adaptive optimal control for continuous-time linear systems with completely unknown dynamics. Automatica 48 (10), 2699 – 2704.

Jonschkowski, R., Brock, O., 2015. Learning state representations with robotic priors. Autonomous Robots 39 (3), 407–428.

Jonschkowski, R., Hafner, R., Scholz, J., Riedmiller, M., 2017. PVEs: Position-velocity encoders for unsupervised learning of structured state representations. In: New Frontiers for Deep Learning in Robotics Workshop at RSS.

Kaelbling, L. P., Littman, M. L., Moore, A. W., 1996. Reinforcement learning: A survey. Journal of Artificial Intelligence Research 4, 237–285.

Kakade, S., 2001. A natural policy gradient. In: Dietterich, T. G., Becker, S., Ghahramani, Z. (Eds.), Advances in Neural Information Processing Systems 14. MIT Press, pp. 1531–1538.

Kakade, S., Langford, J., 2002. Approximately optimal approximate reinforcement learning. In: ICML. Vol. 2. pp. 267–274.

Kamien, M. I., Schwartz, N. L., 2012. Dynamic optimization: the calculus of variations and optimal control in economics and management. Courier Corporation.

Khargonekar, P. P., Dahleh, M. A., 2018. Advancing systems and control research in the era of ml and ai. Annual Reviews in Control.

Kingma, D., Ba, J., 2014. Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

Kober, J., Bagnell, J. A., Peters, J., 2013. Reinforcement learning in robotics: A survey. The International Journal of Robotics Research 32 (11), 1238–1274.

Kocsis, L., Szepesvári, C., 18–22 September 2006. Bandit based Monte-Carlo planning. In: Proceedings 17th European Conference on Machine Learning (ECML-06). Berlin, Germany, pp. 282–293.

Konda, V. R., Tsitsiklis, J. N., 2003. On actor-critic algorithms. SIAM Journal on Control and Optimization 42 (4), 1143–1166.

Koutník, J., Cuccu, G., Schmidhuber, J., Gomez, F., 2013. Evolving large-scale neural networks for vision-based reinforcement learning. In: Proceedings of the 15th annual conference on Genetic and evolutionary computation. ACM, pp. 1061–1068.

Kretchmar, R. M., Young, P. M., Anderson, C. W., Hittle, D. C., Anderson, M. L., Delnero, C. C., Dec 2001. Robust reinforcement learning control with static and dynamic stability. IEEE Transactions on Neural Networks and Learning Systems 11 (15), 1469–1500.

Lagoudakis, M. G., Parr, R., 2003. Least-squares policy iteration. Journal of Machine Learning Research 4, 1107–1149.

Landau, I. D., Lozano, R., M'Saad, M., Karimi, A., 2011. Adaptive Control: Algorithms, Analysis and Applications, 2nd Edition. Communications and Control Engineering. Springer-Verlag London.

Lange, S., Riedmiller, M., Voigtlander, A., 2012. Autonomous reinforcement learning on raw visual input data in a real world application. In: Int. Joint Conf. Neural Networks (IJCNN).

Lazaric, A., Ghavamzadeh, M., Munos, R., 2012. Finite-sample analysis of least-squares policy iteration. Journal of Machine Learning Research 13, 3041–3074.

Lee, A. X., Levine, S., Abbeel, P., 2017. Learning visual servoing with deep features and fitted q-iteration. CoRR abs/1703.11000.
URL `http://arxiv.org/abs/1703.11000`

Levine, S., Finn, C., Darrell, T., Abbeel, P., 2016. End-to-end training of deep visuomotor policies. The Journal of Machine Learning Research 17 (1), 1334–1373.

Levine, S., Koltun, V., 2013. Guided policy search. In: International Conference on Machine Learning. pp. 1–9.

Lewis, F., Liu, D. (Eds.), 2012. Reinforcement Learning and Adaptive Dynamic

Programming for Feedback Control. Wiley.
Lewis, F. L., Vrabie, D., 2009. Reinforcement learning and adaptive dynamic programming for feedback control. IEEE Circuits and Sytems Magazine 9 (3), 32–50.
Lewis, F. L., Vrabie, D., Vamvoudakis, K. G., Dec 2012. Reinforcement learning and feedback control: Using natural decision methods to design optimal adaptive controllers. IEEE Control Systems 32 (6), 76–105.
Li, Y., 2017. Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274.
Liberzon, D., 2011. Calculus of Variations and Optimal Control Theory: A Concise Introduction. Princeton University Press, Princeton, NJ, USA.
Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.
Lin, L.-J., Aug. 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. Machine Learning 8 (3–4), 293–321, special issue on reinforcement learning.
Mann, T. A., Mannor, S., Precup, D., 2015. Approximate value iteration with temporally extended actions. Journal of Artificial Intelligence Research 53, 375–438.
Melo, F. S., Meyn, S. P., Ribeiro, M. I., 5–9 July 2008. An analysis of reinforcement learning with function approximation. In: Proceedings 25th International Conference on Machine Learning (ICML-08). Helsinki, Finland., pp. 664–671.
Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning. In: International Conference on Machine Learning. pp. 1928–1937.
Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D., 2015. Human-level control through deep reinforcement learning. Nature 518, 529–533.
Modares, H., Lewis, F. L., Jiang, Z. P., Oct 2015. H_{infty} tracking control of completely unknown continuous-time systems via off-policy reinforcement learning. IEEE Transactions on Neural Networks and Learning Systems 26 (10), 2550–2562.
Moore, A. W., Atkeson, C. G., 1993. Prioritized sweeping: Reinforcement learning with less data and less time. Machine Learning 13, 103–130.
Moore, K. L., 2012. Iterative learning control for deterministic systems. Springer Science & Business Media.
Munos, R., 2014. From bandits to Monte Carlo tree search: The optimistic principle applied to optimization and planning. Foundations and Trends in Machine Learning 7 (1), 1–130.
Munos, R., Moore, A., 2002. Variable-resolution discretization in optimal control. Machine Learning 49 (2–3), 291–323.
Munos, R., Stepleton, T., Harutyunyan, A., Bellemare, M., 2016. Safe and efficient off-policy reinforcement learning. In: Advances in Neural Information Processing Systems. pp. 1054–1062.
Munos, R., Szepesvári, Cs., 2008. Finite time bounds for fitted value iteration. Journal of Machine Learning Research 9, 815–857.
Olah, C., Mordvintsev, A., Schubert, L., 2017. Feature visualization. Distill 2 (11), e7.
Ormoneit, D., Sen, S., 2002. Kernel-based reinforcement learning. Machine Learning 49 (2–3), 161–178.
Parisotto, E., Ba, J. L., Salakhutdinov, R., 2015. Actor-mimic: Deep multitask and transfer reinforcement learning. arXiv preprint arXiv:1511.06342.
Pazis, J., Lagoudakis, M., 14–18 June 2009. Binary action search for learning continuous-action control policies. In: Proceedings of the 26th International Conference on Machine Learning (ICML-09). Montreal, Canada, pp. 793–800.
Peters, J., Schaal, S., 2008. Natural actor-critic. Neurocomputing 71 (7–9), 1180–1190.
Postoyan, R., Buşoniu, L., Nešić, D., Daafouz, J., 2017. Stability analysis of discrete-time infinite-horizon optimal control with discounted cost. IEEE Transactions on Automatic Control 62 (6), 2736–2749.
Powell, W. B., 2012. Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition. Wiley.
Precup, D., Sutton, R. S., Singh, S. P., 2000. Eligibility traces for off-policy policy evaluation. In: ICML. Citeseer, pp. 759–766.
Puterman, M. L., 1994. Markov Decision Processes—Discrete Stochastic Dynamic Programming. Wiley.
Raghu, M., Poole, B., Kleinberg, J., Ganguli, S., Sohl-Dickstein, J., 2016. On the expressive power of deep neural networks. arXiv preprint arXiv:1606.05336.
Rajeswaran, A., Lowrey, K., Todorov, E. V., Kakade, S. M., 2017. Towards generalization and simplicity in continuous control. In: Advances in Neural Information Processing Systems. pp. 6553–6564.
Recht, B., Re, C., Wright, S., Niu, F., 2011. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In: Advances in neural information processing systems. pp. 693–701.
Riedmiller, M., 2005. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In: European Conference on Machine Learning. Springer, pp. 317–328.
Ross, S., Pineau, J., Chaib-draa, B., Kreitmann, P., Jul. 2011. A bayesian approach for learning and planning in partially observable markov decision processes. Journal of Machine Learning Research 12, 1729–1770.
Rummery, G. A., Niranjan, M., September 1994. On-line Q-learning using connectionist systems. Tech. Rep. CUED/F-INFENG/TR166, Engineering Department, Cambridge University, UK, available at http://mi.eng.cam.ac.uk/reports/svr-ftp/rummery_tr166.ps.Z.
Russell, S. J., Norvig, P., 2016. Artificial Intelligence: A Modern Approach, 3rd Edition. Pearson Education.
Salimans, T., Kingma, D. P., 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In: Advances in Neural Information Processing Systems. pp. 901–909.
Schaul, T., Horgan, D., Gregor, K., Silver, D., 2015. Universal value function approximators. In: International Conference on Machine Learning. pp. 1312–1320.
Schaul, T., Quan, J., Antonoglou, I., Silver, D., 2016. Prioritized experience replay. In: International Conference on Learning Representations (ICLR).
Scherrer, B., 21–24 June 2010. Should one compute the Temporal Difference fix point or minimize the Bellman Residual? the unified oblique projection view. In: Proceedings 27th International Conference on Machine Learning (ICML-10). Haifa, Israel, pp. 959–966.
Schmidhuber, J., 1991. Adaptive confidence and adaptive curiosity. Tech. rep., Arcisstr. 21, 800 Munchen 2.
Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015a. Trust region policy optimization. In: International Conference on Machine Learning. pp. 1889–1897.
Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P., 2015b. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438.
Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.
Shelhamer, E., Mahmoudieh, P., Argus, M., Darrell, T., 2016. Loss is its own reward: Self-supervision for reinforcement learning. arXiv:1612.07307.
Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al., 2016. Mastering the game of go with deep neural networks and tree search. nature 529 (7587), 484–489.
Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms. In: ICML.
Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., Hassabis, D., 2017. Mastering the game of go without human knowledge. Nature 550, 354–359.
Singh, S., Jaakkola, T., Littman, M. L., Szepesvári, Cs., 2000. Convergence results for single-step on-policy reinforcement-learning algorithms. Machine Learning 38 (3), 287–308.
Singh, S., Sutton, R., 1996. Reinforcement learning with replacing eligibility traces. Machine Learning 22 (1–3), 123–158.
Sutton, R. S., Barto, A. G., 1998. Reinforcement Learning: An Introduction. MIT Press.
Sutton, R. S., Barto, A. G., 2018. Reinforcement Learning: An Introduction, 2nd Edition. MIT Press.
Sutton, R. S., Mahmood, A. R., White, M., 2016. An emphatic approach to the problem of off-policy temporal-difference learning. The Journal of Machine Learning Research 17 (1), 2603–2631.
Sutton, R. S., McAllester, D., Singh, S., Mansour, Y., 2000. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: Advances in Neural Information Processing Systems 12. MIT Press, pp. 1057–

1063.

Szepesvári, Cs., 2010. Algorithms for Reinforcement Learning. Morgan & Claypool Publishers.

Thiery, C., Scherrer, B., 21–24 June 2010. Least-squares λ policy iteration: Bias-variance trade-off in control problems. In: Proceedings 27th International Conference on Machine Learning (ICML-10). Haifa, Israel, pp. 1071–1078.

Thrun, S. B., 1992. Efficient exploration in reinforcement learning. Tech. rep., Pittsburgh, PA, USA.

Tieleman, T., Hinton, G., 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4 (2), 26–31.

Tolić, D., Fierro, R., Ferrari, S., 2012. Optimal self-triggering for nonlinear systems via approximate dynamic programming. In: IEEE Multi-Conference on Systems and Control. pp. 879–884.

Tolić, D., Palunko, I., 2017. Learning suboptimal broadcasting intervals in multi-agent systems. IFAC-PapersOnLine 50 (1), 4144 – 4149, 20th IFAC World Congress.

Tsitsiklis, J. N., Van Roy, B., 1997. An analysis of temporal difference learning with function approximation. IEEE Transactions on Automatic Control 42 (5), 674–690.

Tucker, G., Bhupatiraju, S., Gu, S., Turner, R. E., Ghahramani, Z., Levine, S., 2018. The mirage of action-dependent baselines in reinforcement learning. arXiv preprint arXiv:1802.10031.

van Hasselt, H., 2010. Double q-learning. In: Advances in Neural Information Processing Systems. pp. 2613–2621.

van Hasselt, H., Guez, A., Hessel, M., Mnih, V., Silver, D., 2016a. Learning values across many orders of magnitude. In: Neural Information Processing Systems (NIPS).

van Hasselt, H., Guez, A., Silver, D., 2016b. Deep reinforcement learning with double q-learning. In: Conf. Artificial Intelligence (AAAI).

Wang, Y., O'Donoghue, B., Boyd, S., 2014. Approximate dynamic programming via iterated bellman inequalities. International Journal of Robust and Nonlinear Control 25 (10), 1472–1496.

Wang, Z., Bapst, V., Heess, N., Mnih, V., Munos, R., Kavukcuoglu, K., de Freitas, N., 2016. Sample efficient actor-critic with experience replay. arXiv preprint arXiv:1611.01224.

Watkins, C. J. C. H., 1989. Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge.

Watkins, C. J. C. H., Dayan, P., 1992. Q-learning. Machine Learning 8, 279–292.

Watter, M., Springenberg, J., Boedecker, J., Riedmiller, M., 2015. Embed to control: A locally linear latent dynamics model for control from raw images. In: Neural Information Processing Systems (NIPS).

Weinstein, A., Littman, M. L., 25–19 June 2012. Bandit-based planning and learning in continuous-action Markov decision processes. In: Proceedings 22nd International Conference on Automated Planning and Scheduling (ICAPS-12). São Paulo, Brazil.

Wiering, M., van Otterlo, M. (Eds.), 2012. Reinforcement Learning: State of the Art. Vol. 12. Springer.

Williams, R. J., 1992. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning 8, 229–256.

Williams, R. J., Peng, J., 1991. Function optimization using connectionist reinforcement learning algorithms. Connection Science 3 (3), 241–268.

Wiskott, L., Sejnowski, T. J., 2002. Slow feature analysis: Unsupervised learning of invariances. Neural Computation 14 (4), 715–770.

Yang, X., Liu, D., Wei, Q., 2014. Online approximate optimal control for affine non-linear systems with unknown internal dynamics using adaptive dynamic programming. IET Control Theory Applications 8 (16), 1676–1688.

Yu, H., Bertsekas, D. P., 2009. Convergence results for some temporal difference methods based on least squares. IEEE Transactions on Automatic Control 54 (7), 1515–1531.

Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O., 2016. Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530.

Zhang, H., Cui, L., Zhang, X., Luo, Y., Dec 2011. Data-driven robust approximate optimal tracking control for unknown general nonlinear systems using adaptive dynamic programming method. IEEE Transactions on Neural Networks 22 (12), 2226–2236.

Zhang, J., Springenberg, J. T., Boedecker, J., Burgard, W., 2017. Deep reinforcement learning with successor features for navigation across similar environments. In: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on. IEEE, pp. 2371–2378.