Experience Replay for Real-Time Reinforcement Learning Control

Sander Adam, Lucian Buşoniu, and Robert Babuška

***Abstract*—Reinforcement learning (RL) algorithms can automatically learn optimal control strategies for nonlinear, possibly stochastic systems. A promising approach for RL control is experience replay (ER), which quickly learns from a limited amount of data by repeatedly presenting these data to an underlying RL algorithm. Despite its benefits, ER RL has been studied only sporadically in the literature, and its applications have largely been confined to simulated systems. Therefore, in this paper we evaluate ER RL on real-time control experiments involving a pendulum swing-up problem and the vision-based control of a goalkeeper robot. These real-time experiments are complemented by simulation studies and comparisons with traditional RL. As a preliminary, we develop a general ER framework that can be combined with essentially any incremental RL technique, and instantiate this framework for the approximate Q-learning and SARSA algorithms. The successful real-time learning results presented here are highly encouraging for the applicability of ER RL in practice.**

***Index Terms*—reinforcement learning, experience replay, real-time control, robotics, Q-learning, SARSA.**

I. Introduction

Reinforcement learning (RL) can be applied to problems for which it is difficult or impossible to design the controller in advance, for instance because a process model is not available. This technique pertains to a broad class of dynamic systems, including nonlinear deterministic and stochastic processes [3], [32] and is therefore applicable in many domains [2], [12], [13], [24]. Compact, approximate representations of the solution [5], [6] are needed in control problems, to deal with the continuous variables appearing in such problems. In practical real-time applications, algorithms for online approximate RL must satisfy two somewhat conflicting requirements. First, they need to be *data efficient* to achieve acceptable performance after only a brief interaction with the system. A controller that performs poorly for a long period will never be accepted in practice, even if it is believed to eventually yield an optimal strategy. Second, online RL algorithms must be *computationally efficient*, so that they work in real time. Most online RL approaches are based on the classical Q-learning [36] and SARSA [25] algorithms, which are indeed computationally efficient, but are data inefficient: they use every sample once, to incrementally improve the solution, after which they discard the sample, see e.g., [20], [27], [34].

A promising approach for practical RL is *experience replay (ER)*, in which the data acquired during the online learning process are stored and presented repeatedly to the underlying RL algorithm. This increases data efficiency, while exploiting the computational efficiency of the underlying algorithm. ER was introduced by Lin [19], in the context of RL with neural network approximation. Later on, [16] illustrated using simulations that reusing data in Q-learning leads to a performance similar to that of batch RL algorithms, and [37] showed that ER can be added to actor-critic RL algorithms without endangering their convergence properties. Despite such encouraging results, ER has been studied only sporadically in the RL literature, and only a few real-time applications have been presented [9], [29]. In fact, applications of RL have so far largely been restricted to simulated systems. In order to improve the visibility of this field and to stimulate real-world applications, it is necessary to go beyond simulations and demonstrate that RL methods can be effectively used in the real-time control of physical systems.

Therefore, in this paper we contribute a thorough experimental evaluation of ER RL, which includes real-time results. We first introduce a general ER framework, which can be combined with any incremental RL algorithm. This framework is instantiated for approximate Q-learning and SARSA, yielding ER-Q-learning and ER-SARSA. We then evaluate these ER algorithms on a series of increasingly complex problems. We first consider simulation studies of the well-known pendulum swing-up and of robotic manipulator control. In the swing-up problem, the ER algorithms are compared with the original approximate Q-learning and SARSA, as well as with a batch RL method that we augment with online data collection. We present real-time experimental results on two physical systems: the real inverted pendulum and a robotic soccer goalkeeper.

An approach related to ER is to indirectly reuse the data, by first building a model of the system and then exploiting it to generate new data. This is done in the so-called Dyna or model-learning approaches [21], [31], [33]. Building a model will generally take less memory than storing the raw data, and can diminish the effects of noise by 'filtering' the data into the model parameters. However, model learning will incur additional computational costs and, more importantly, will introduce modeling errors that can significantly decrease performance [33], a problem that does not affect ER. A different way to look at ER is as a bridge between incremental and batch RL algorithms [11], [17], allowing for tradeoffs that are not possible with either type of algorithm. We finally note a related thread of research that exploits the data efficiency of batch algorithms in an online setting [7], [18].

The remainder of the paper is organized as follows. Section II presents the necessary background in exact and approximate RL. In Section III, we introduce and discuss our ER framework. In Section IV, the performance of the ER algorithms is evaluated in an extensive simulation and experimental study. Section V concludes the paper.

S. Adam is with the Large Corporates & Merchant Banking division of ABN AMRO Bank in the Netherlands (mail@ajadam.nl). L. Buşoniu and R. Babuška are with the Delft Center for Systems and Control, TUDelft, the Netherlands (email: i.l.busoniu@tudelft.nl, r.babuska@tudelft.nl). This research was partly supported by the ICIS project (grant #BSIK03024).

II. Reinforcement learning preliminaries

This section first introduces the RL problem in the framework of Markov decision processes (MDPs). Then, exact and approximate variants of the Q-learning and SARSA algorithms are described.

A. Markov decision processes

An MDP is defined by its state space X, its action space U, its transition probability function $f : X \times U \times X \to [0,\infty)$, and its reward function $\rho : X \times U \times X \to \mathbb{R}$. At each discrete time step k, given the state x_k, the controller takes an action u_k according to a control policy $h : X \to U$. The probability that the next state x_{k+1} belongs to a region $X_{k+1} \subset X$ of the state space is then $\int_{X_{k+1}} f(x_k,u_k,x')\mathrm{d}x'$. For any x and u, $f(x,u,\cdot)$ is assumed to define a valid density of the argument "$\cdot$". After the transition to x_{k+1}, a reward r_{k+1} is provided according to the reward function ρ: $r_{k+1} = \rho(x_k,u_k,x_{k+1})$. For deterministic MDPs, the transition probability function f is replaced by the transition function, $\overline{f} : X \times U \to X$, and the reward is completely determined by the current state and action: $r_{k+1} = \overline{\rho}(x_k,u_k)$, $\overline{\rho} : X \times U \to \mathbb{R}$.

The expected infinite-horizon discounted return for an initial state x_0 under a policy h is:[1]

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

where $\gamma \in [0,1)$ is the discount factor. The notation $a \sim p(\cdot)$ means that the random variable a is drawn from the density p. The goal is to find an optimal policy h^* that maximizes the expected return (1) for every initial state x_0. So, the long-term performance (return) must be maximized using only feedback about the immediate, one-step performance (reward). For any MDP, there exists a deterministic optimal policy.

Every policy h is characterized by its state-action value function (Q-function), $Q^h : X \times U \to \mathbb{R}$, which gives the return when starting in a given state, applying a given action, and following h thereafter. For any h, Q^h is unique and can be found by solving the *Bellman equation*:

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

The optimal Q-function is defined as $Q^*(x,u) = \max_h Q^h(x,u)$, and satisfies the *Bellman optimality equation*:

$$Q^*(x,u) = \mathrm{E}_{x'\sim f(x,u,\cdot)} \left\{ \rho(x,u,x') + \gamma \max_{u'} Q^*(x',u') \right\} \quad (3)$$

A policy h^* that selects for every state an action with the best optimal Q-value, i.e., that satisfies:

$$h^*(x) \in \arg\max_u Q^*(x,u) \quad (4)$$

is optimal (it maximizes the return). A policy that maximizes a Q-function in this way is said to be *greedy* in that Q-function. So, finding an optimal policy can be done by finding Q^* and taking actions with a greedy policy in Q^*.

[1] We assume that the MDP and the policies h have suitable properties such that the expected return and the Bellman equations in the remainder of this section are well-defined. See, e.g., Ch. 9 of [4] and App. A of [3] for a discussion of these properties.

B. Exact reinforcement learning

Two classical algorithms for discrete RL problems are Q-learning [36] and SARSA [25]. They are online and model-free, i.e., they estimate a solution while actually controlling the process, without requiring the knowledge of the transition and reward functions f and ρ. The Q-learning algorithm starts with an arbitrary Q-function Q_0, observes transitions $(x_k,u_k,r_{k+1},x_{k+1})$, and after each transition updates the Q-function with:

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k [r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u') - Q_k(x_k,u_k)] \quad (5)$$

where $\alpha_k \in (0,1)$ is the learning rate. The term between square brackets is the temporal difference, i.e., the difference between the current estimate $Q_k(x_k,u_k)$ of the optimal Q-value of (x_k,u_k) and the updated estimate $r_{k+1} + \gamma \max_{u'} Q_k(x_{k+1},u')$. This updated estimate is in fact a sample of the random quantity on the right hand side of the Bellman equation (3), computed for the state-action pair (x_k,u_k). The Q-learning temporal difference at time step k is denoted by δ_k^Q.

The SARSA algorithm starts with an arbitrary Q-function Q_0 and updates it using tuples $(x_k,u_k,r_{k+1},x_{k+1},u_{k+1})$, as follows:

$$Q_{k+1}(x_k,u_k) = Q_k(x_k,u_k) + \alpha_k [r_{k+1} + \gamma Q_k(x_{k+1},u_{k+1}) - Q_k(x_k,u_k)] \quad (6)$$

The term between square brackets is the temporal difference between the current estimate $Q_k(x_k,u_k)$ of the Q-value of (x_k,u_k) *under the current policy*, and the updated estimate $r_{k+1} + \gamma Q_k(x_{k+1},u_{k+1})$, which is a sample of the right hand side of the Bellman equation (2). The SARSA temporal difference is denoted by δ_k^S.

Q-learning is off-policy: regardless of the policy being followed, it always estimates the optimal Q-function. In contrast, SARSA is on-policy: at every update, it aims to estimate the Q-function of the policy being followed.

Q-learning converges to Q^* as $k \to \infty$ if $\sum_{k=0}^{\infty} \alpha_k^2$ is finite, $\sum_{k=0}^{\infty} \alpha_k$ is infinite, and all the state-action pairs are visited infinitely often as the number of transitions approaches infinity [14], [36]. The latter condition can be satisfied if, among others, the probability of choosing any action is non-zero in every encountered state; this is called exploration. The controller also has to exploit its current knowledge in order to perform well, e.g., by selecting greedy actions in the current Q-function. A classical way to balance exploration with exploitation is the ε-greedy policy:

$$u_k \leftarrow \begin{cases} u \in \arg\max_{\bar{u}} Q(x_k,\bar{u}) & \text{w.p. } 1-\varepsilon_k \\ \text{a uniform random action in } U & \text{w.p. } \varepsilon_k \end{cases} \quad (7)$$

where "w.p." stands for "with probability", and $\varepsilon_k \in (0,1)$ is the exploration probability. In order to converge to Q^*, in addition to the conditions required by Q-learning, SARSA further requires that the policy being followed converges to the greedy policy. This can be satisfied by using (7) with a decreasing ε_k, such that $\lim_{k\to\infty} \varepsilon_k = 0$.

C. Approximate reinforcement learning

The algorithms of Section II-B require that the Q-functions are represented exactly. When the state or action spaces of the MDP contain a large or infinite number of elements, Q-functions cannot be represented exactly, but must be approximated. We consider linearly parameterized approximators:

$$\hat{Q}(x,u) = \sum_{i=1}^{n} \phi_i(x,u)\theta_i = \phi^{\mathrm{T}}(x,u)\theta \tag{8}$$

where $\theta \in \mathbb{R}^n$ is the parameter vector and $\phi : X \times U \to \mathbb{R}^n$ is the vector of basis functions (BFs), $[\phi_1(x,u),\ldots,\phi_n(x,u)]^{\mathrm{T}}$. Such representations simplify the theoretical analysis of the resulting algorithms, see, e.g., [5] and Ch. 6 of [3]. Hence, they are widely used in approximate RL, under various names such as "interpolative representations" [34], "soft aggregation" [27], and "fuzzy approximation" [15].

Q-learning and SARSA can be combined with linear approximation by using gradient-based updates. Such a gradient-based algorithm updates the parameters with:

$$\theta_{k+1} = \theta_k + \alpha_k \hat{\delta}_k \frac{\partial}{\partial \theta_k} \hat{Q}_k(x_k,u_k) = \theta_k + \alpha_k \hat{\delta}_k \phi(x_k,u_k) \tag{9}$$

where $\hat{\delta}_k$ is a generic approximate temporal difference. To obtain approximate Q-learning, $\hat{\delta}_k$ is replaced by the following approximation of the Q-learning temporal difference:

$$\hat{\delta}_k^Q = r_{k+1} + \gamma \max_{u'} \phi(x_{k+1},u')^{\mathrm{T}}\theta_k - \phi(x_k,u_k)^{\mathrm{T}}\theta_k$$

Similarly, to obtain approximate SARSA, $\hat{\delta}_k$ is replaced by:

$$\hat{\delta}_k^S = r_{k+1} + \gamma \phi(x_{k+1},u_{k+1})^{\mathrm{T}}\theta_k - \phi(x_k,u_k)^{\mathrm{T}}\theta_k$$

Approximate Q-learning does not converge in general [1], but it does converge (under mild assumptions on the approximator) if the policy used to choose actions is fixed [20], [34]. [20] showed that SARSA converges w.p. 1 to a fixed point, if the dependence of the policy on the parameter vector satisfies a certain Lipschitz continuity condition.

The algorithms above only use the latest transition in every update. Learning can be sped up by using the fact that this latest transition is the causal result of an entire trajectory. To this end, recently visited state-action pairs are made eligible for updating by using an *eligibility trace* $e \in \mathbb{R}^n$. In this paper, we use replacing traces [26], which are initialized to 0 and updated with:

$$e_k = \min(\lambda e_{k-1} + \phi(x_k,u_k), 1) \tag{10}$$

where $\lambda \in [0,1)$ is the trace decay rate and the minimum operation is applied element-wise. Eligibility traces are used in Q-learning and SARSA by changing the update (9) to:

$$\theta_{k+1} = \theta_k + \alpha_k \hat{\delta}_k e_k \tag{11}$$

For $\lambda = 0$, the original algorithms are recovered. Note that in Q-learning, whenever an exploratory action is taken, the causality of the sequence of state-action pairs is broken and the eligibility trace should be reset to 0. This is not the case for SARSA, because SARSA always estimates the Q-function of the current policy, which includes the exploration.

III. Experience replay

This section presents our approach to using *experience replay (ER)* with reinforcement learning. The proposed ER procedure stores transition samples and repeatedly presents them to a gradient-based, incremental RL algorithm. Thus, the computational efficiency of the underlying gradient-based algorithm is exploited, while a high data efficiency is achieved by reusing the samples. As approximation is necessary in most control problems (and also in most realistic RL problems outside the field of control), we only consider the approximate case, in particular, linearly parameterized Q-functions (8).

A. ER framework

Algorithm 1 presents the general ER framework. At every time step k, a transition sample $(x_k,u_k,r_{k+1},x_{k+1})$ is observed and stored in the sample database D (lines 5–8). To choose actions, a greedy policy in the current Q-function is used together with exploration (line 5). In this paper, the classical ε-greedy exploration strategy (7) is used; many other strategies are possible. Once every T steps, the data in the sample database are used N times to update the Q-function parameters θ with the underlying RL algorithm (line 11). This implicitly performs a large, one-step update of the policy. In the interval between consecutive updates, the approximate Q-function and therefore the greedy policy remain constant. We use the term *trajectory* to refer to every sequence of T samples collected in-between two consecutive updates. Trajectories are distinguished by their index l, and discrete time within a trajectory is denoted by $\tau = 0,\ldots,T-1$. Note that Algorithm 1 can easily use a time-varying learning rate and exploration probability.

Next, we instantiate Algorithm 1 using approximate Q-learning and SARSA. For each of the two algorithms, we fur-

Algorithm 1 Experience replay RL

Input: underlying RL algorithm LEARN,
number of replays N, length of each trajectory T,
BFs $\phi_1,\ldots,\phi_n$, discount factor γ, learning rate α,
exploration probability ε

1: initialize θ arbitrarily (e.g., identically 0)
2: $D \leftarrow \emptyset$; $l \leftarrow 1$; $k \leftarrow 0$
3: observe initial state x_0
4: **for** every time step k **do**
5: $\quad u_k \leftarrow \begin{cases} u \in \arg\max_{\bar{u}} \phi^{\mathrm{T}}(x_k,\bar{u})\theta & \text{w.p. } 1-\varepsilon \\ \text{a uniform random action in } U & \text{w.p. } \varepsilon \end{cases}$
6: $\quad$ apply u_k, observe resulting x_{k+1} and reward r_{k+1}
7: $\quad$ compute transition index within current trajectory: $\tau \leftarrow k-(l-1)T$
8: $\quad$ add transition sample to the database: $D \leftarrow D \cup \{(k,l,\tau,x_k,u_k,x_{k+1},r_{k+1})\}$
9: $\quad k \leftarrow k+1$
10: $\quad$ **if** $k = lT$ (a T-step trajectory was collected) **then**
11: $\quad\quad$ update Q-function parameters: $\theta \leftarrow \text{LEARN}(\theta,D,N,T,l,\alpha,\gamma)$
12: $\quad\quad l \leftarrow l+1$
13: $\quad$ **end if**
14: **end for**

Algorithm 2 Q-LEARN-SAMPLES$(\theta, D, N, T, l, \alpha, \gamma)$

loop NlT times
  retrieve a random sample $(k, l', \tau, x, u, x', r)$ from D, using a uniform distribution
  $\theta \leftarrow \theta + \alpha[r + \gamma \max_{u'} \phi(x', u')^{\mathrm{T}}\theta - \phi(x,u)^{\mathrm{T}}\theta] \cdot \phi(x,u)$
end loop
Output: θ

Algorithm 3 Q-LEARN-TRAJECTORIES$(\theta, D, N, T, l, \alpha, \gamma)$

loop Nl times
  select a random trajectory l' using a uniform distribution over $\{1, \ldots, l\}$
  for $\tau = 0, \ldots, T-1$ **do**
    retrieve from D the sample $(k, l', \tau, x, u, x', r)$ corresponding to l' and τ
    $\theta \leftarrow \theta +$
      $\alpha[r + \gamma \max_{u'} \phi(x', u')^{\mathrm{T}}\theta - \phi(x,u)^{\mathrm{T}}\theta] \cdot \phi(x,u)$
  end for
end loop
Output: θ

ther differentiate two ways of reusing the samples. In the first way, samples are independently and randomly chosen from D. In the second way, trajectories are randomly chosen from D, and for every chosen trajectory, the samples are presented in their causal order. So, a total of four algorithms is obtained: (i) ER-Q-learning on separate samples, (ii) ER-Q-learning on trajectories, (iii) ER-SARSA on separate samples, and (iv) ER-SARSA on trajectories. For instance, to obtain ER-Q-learning on separate samples, the generic procedure LEARN of Algorithm 1 is replaced by the procedure Q-LEARN-SAMPLES from Algorithm 2. To obtain ER-Q-learning on trajectories, Q-LEARN-TRAJECTORIES (Algorithm 3) should be used instead. ER-SARSA on separate samples and trajectories can be obtained in an entirely similar fashion.

Recall from Section II-C that approximate Q-learning may not converge in general, but can be guaranteed to converge when it employs a fixed policy [20]. While the ER-Q-learning policy is not fixed throughout the learning process, it is fixed along the intervals between consecutive updates, which may increase the stability of ER-Q-learning compared to the original Q-learning. A useful special case is identified in Section 6.5 of [10]: for a piecewise constant Q-function approximator, the ER procedure in Algorithm 2 asymptotically converges to a well-defined solution, given by a certain model-learning algorithm based on the same samples. Approximate SARSA converges—under appropriate conditions—both when the policy is changed after every sample [20], and when it is changed only after a large number of updates [23]. This indicates that the convergence properties of ER-SARSA should be similar to those of SARSA. Note that convergence often relies on processing the samples in the order they are obtained from the system, which points toward using ER on trajectories. Nevertheless, replaying samples in a different order may propagate information more efficiently, see also Section III-B below.[2]

An important feature of these ER algorithms is their ability to trade off computational complexity for learning speed. This is achieved by tuning N, the number of experience replays at every update. A low value of N increases the computational efficiency, while a high value of N accelerates learning. In order to best exploit the available data, we have chosen to perform, at every update stage, a number of updates that is N times the size of the sample database. However, this is in general not a requirement, and the algorithms could also perform a fixed number of updates, which would allow a finer-grained control of their computational expenses during the update stages. With the exception of possible slower learning, such an algorithm should perform similarly to Algorithm 1.

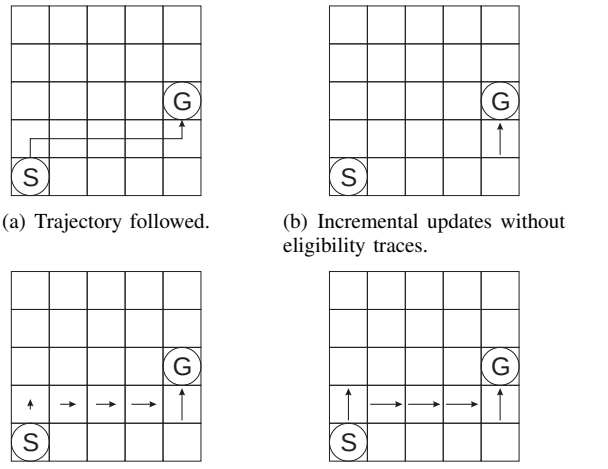
If the current update scheme is used, the sample database must be prevented from growing indefinitely by periodical pruning, which could be done e.g., by removing old samples or by clustering the samples into representative prototypes. The former option is more suitable if the system is slowly changing, as then old transition samples are less representative of the current behavior of the system. The experiments we present in the sequel are short enough to not require pruning. Finally, note that Algorithm 1 can easily be extended to work in episodic problems, i.e., problems in which the process eventually reaches a terminal state that it can no longer leave. An episode is a trajectory started in any state and ending in a terminal state. The variable-length episodes can be used instead of fixed-length trajectories in Algorithm 1.

B. Indirect benefits of ER

While the efficient reutilization of data is its central motivation, ER also provides other, less obvious benefits. To illustrate these benefits, we use the simple maze problem shown in Figure 1(a), in which an agent must find the shortest path from the start state S to the goal state G, while only receiving a positive reward upon reaching the goal state; all the other rewards are zero. Although the maze problem has discrete variables, the analysis extends directly to the approximate, continuous-variable case.

One benefit of ER is that it asymptotically has similar effects with eligibility traces, because it propagates similar information to that transmitted by the eligibility traces. Consider that the agent has traveled the trajectory in Figure 1(a), and that it learns with an incremental algorithm relying on temporal differences, e.g., Q-learning or SARSA. To propagate the maximum amount of information, set the learning rate to $\alpha = 1$. If no eligibility traces are used, the reward received upon reaching G is propagated only to the state just before G, see Figure 1(b). If eligibility traces with $\lambda < 1$ are used, the reward is propagated to all the states along the trajectory, but with an eligibility that decreases exponentially while moving back along the trajectory, see Figure 1(c). When $\lambda = 1$ is used, the reward fully propagates to all the states along the trajectory (properly discounted, of course), see Figure 1(d).

[2]This entire analysis is only cautiously suggested because, so far, all convergence proofs rely on *sufficient* conditions, and any or all of these conditions may, in fact, not be *necessary*.



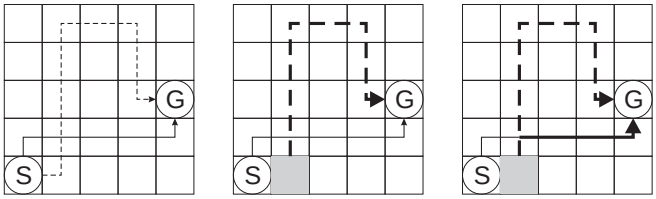
(a) Trajectory followed.

(b) Incremental updates without eligibility traces.

(c) Incremental updates with eligibility traces, $\lambda < 1$.

(d) ER updates; and incremental updates with eligibility traces, $\lambda = 1$.

Fig. 1. ER can replace eligibility traces. The length of the arrows symbolizes the amount of information propagated from the goal to each state along the trajectory.



(a) Trajectories.

(b) Incremental updates with eligibility traces.

(c) ER updates.

Fig. 2. ER aggregates information from multiple trajectories. The gray square receives information from the squares crossed by thick lines.

Consider now an ER algorithm, which replays the samples along the trajectory several times. Sometimes, samples earlier along the trajectory will be replayed *after* samples later in the trajectory, which will propagate information back along the trajectory despite the lack of an eligibility trace. As the number of replays increases, the result becomes similar to that obtained using eligibility traces with $\lambda = 1$. This feature of ER is helpful because the user gets the benefits of an eligibility trace without having to tune the additional parameter λ. A similar effect was analyzed in [8].

Another beneficial effect in ER is the aggregation of information from multiple trajectories. Consider again the maze example, and assume the agent has traveled the two trajectories in Figure 2(a). If an incremental algorithm with eligibility traces is used, only the rewards encountered over the dashed trajectory can be propagated back to the state in the gray square, see Figure 2(b). In contrast, with an ER algorithm, which interleaves samples from both trajectories, the state in the gray square benefits from information along *both* trajectories, see Figure 2(c).

C. Time complexity of ER algorithms

The most computationally complex (in an asymptotic sense) component of the ER algorithms is the update of the parameter vector, shown at line 11 of Algorithm 1. The routine operations of computing the policy and saving the samples are cheaper and will be ignored in the remainder of this section. Consider a single update of the parameter vector with the Q-learning algorithm:

$$\theta \leftarrow \theta + \alpha[r + \gamma \max_{u'} \phi(x', u')^{\mathrm{T}}\theta - \phi(x,u)^{\mathrm{T}}\theta] \cdot \phi(x,u)$$

Here we only consider discrete-action approximation, in which the action space U is discretized into a small number of values $U_{\mathrm{d}} = \{u_1, \ldots, u_M\}$. In this case, the maximization in the Q-learning update can be solved by enumeration over the discrete actions,[3] so its complexity is $O(n)$. The complexity of the entire Q-learning update is also $O(n)$. While the SARSA update does not contain a maximization, its asymptotic complexity is nevertheless also $O(n)$. After each trajectory l, a number of NlT samples are retrieved from the database and used to incrementally update the parameter vector. Together, these updates have a complexity of $O(nNlT)$. This is true whether separate samples or entire trajectories are replayed. The entire procedure is repeated as l increases to some maximum value L. The total complexity is:

$$O(nNL^2T) \tag{12}$$

This complexity holds for all the four ER algorithms considered (ER-Q-learning on separate samples or trajectories, and ER-SARSA on separate samples or trajectories).

IV. Experimental and simulation studies

In the sequel, extensive simulation and real-life experiments are carried out to assess the performance of the ER algorithms: they are used to swing up an inverted pendulum in Section IV-A, to stabilize a robotic manipulator in Section IV-B, and to control a robotic goalkeeper in Section IV-C.

A. Inverted pendulum swing-up

The first example involves the swing-up of an underactuated, inverted pendulum. First, using simulation, ER-Q-learning and ER-SARSA are compared with the classical Q-learning and SARSA algorithms, as well as with a batch RL algorithm. Then, the ER algorithms are applied to the real pendulum system.

1) Inverted pendulum swing-up problem: The inverted pendulum consists of a weight attached to a disk that is actuated by a DC motor and rotates in a vertical plane, see Figure 3. The pendulum must be stabilized in the unstable equilibrium (pointing up), but the motor's power is insufficient to push the pendulum up in a single rotation from every initial state. Instead, from certain states (e.g., pointing down), the pendulum needs to be swung back and forth (destabilized) to gather energy, prior to being pushed up and stabilized.

The dynamics of the pendulum are:

$$\ddot{\vartheta} = \left(mgl\sin(\vartheta) - b\dot{\vartheta} - K^2\dot{\vartheta}/R + Ku/R\right)/J$$

where $J = 1.91 \cdot 10^{-4}\,\mathrm{kgm}^2$, $m = 0.055\,\mathrm{kg}$, $g = 9.81\,\mathrm{m/s}^2$, $l = 0.042\,\mathrm{m}$, $b = 3 \cdot 10^{-6}\,\mathrm{Nms/rad}$, $K = 0.0536\,\mathrm{Nm/A}$, $R =$

[3]For continuous-action Q-function approximators, the maximization over the action variable is a potentially difficult nonlinear optimization problem. The cost of solving it is dependent on the particular approximator considered.

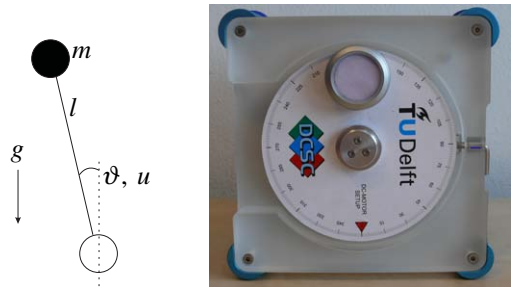


Fig. 3. A schematic representation of the inverted pendulum (left), and the real system (right).

9.5 Ω. The state vector is $x = [\vartheta, \dot{\vartheta}]^{\mathrm{T}}$. The angle ϑ varies in the interval $[-\pi, \pi)$ rad, with $\vartheta = 0$ pointing up, and "wraps around" so that, e.g., a rotation of $3\pi/2$ corresponds to $\vartheta = -\pi/2$. The control action u is constrained to $[-3, 3]$ V, and the velocity $\dot{\vartheta}$ is restricted to $[-15\pi, 15\pi]$ rad/s, using saturation. The sampling time is $T_{\mathrm{s}} = 0.005$ s. The stabilization goal is expressed by the reward function:

$$\overline{\rho}(x,u) = -x^{\mathrm{T}} Q_{\mathrm{rew}} x - R_{\mathrm{rew}} u^2,$$
$$\text{with } Q_{\mathrm{rew}} = \mathrm{diag}[5, 0.1],\ R_{\mathrm{rew}} = 1$$

with a discount factor of $\gamma = 0.98$, which is sufficiently large to obtain a good control policy.

To approximate the Q-function, an equidistant 11×11 grid of Gaussian radial BFs (RBFs) is defined over the state space, and the action space is discretized into 3 discrete values: $U_{\mathrm{d}} = \{-3, 0, 3\}$. The RBFs are normalized, axis-parallel, and have identical radii. The value of the RBF radius along each dimension is identical to the distance between two adjacent RBFs along that dimension (the grid step). To obtain the $n = 3 \cdot 11^2 = 363$ state-action BFs, the RBFs are replicated for every discrete action, and all the BFs that do not correspond to the current discrete action are taken equal to 0. So, if the vector of RBFs is $\bar{\phi}(x) = [\bar{\phi}_1(x), \ldots, \bar{\phi}_{121}(x)]^{\mathrm{T}}$, then the vector of state-action BFs is $\phi(x,u) = [\mathscr{I}(u = -3) \cdot \bar{\phi}^{\mathrm{T}}(x), \mathscr{I}(u = 0) \cdot \bar{\phi}^{\mathrm{T}}(x), \mathscr{I}(u = 3) \cdot \bar{\phi}^{\mathrm{T}}(x)]^{\mathrm{T}}$, where the indicator function $\mathscr{I}$ is 1 when its argument is true, and 0 otherwise.

After each experiment is completed, snapshots of the current policy at increasing moments of time are evaluated. This produces a curve recording the performance of the policy as it evolves over time. During performance evaluation, learning and exploration are turned off. Policies are evaluated by estimating their average return over a set X_0 of representative initial states. The discounted infinite-horizon return of any state is approximated by simulating a sufficiently long, but finite trajectory.

2) Simulation results. Comparison with Q-learning, SARSA, and LSPI: To apply the ER algorithms, trajectories with a length of 1.5 s are used, leading to $T = 300$ samples along each trajectory. Each trajectory (learning trial) is started in a random state chosen from a uniform distribution over X. At the end of each trajectory, real-time control is temporarily suspended while all the available samples are presented to the underlying RL algorithm $N = 10$ times, using a fixed learning rate $\alpha = 0.1$. The exploration probability is initialized at 1, is kept constant along each trajectory, and decays exponentially with a rate of 0.9886 at the end of each trajectory, so that after 200 trajectories, the exploration rate is 0.1.[4] The learning performance is evaluated using the set of initial states $X_0 = \{-\pi, -\pi/2, 0, \pi/2\} \times \{-10\pi, -5\pi, , -2\pi, -\pi, 0, \pi, 2\pi, 5\pi, 10\pi\}$, which evenly covers the state space in order to obtain a representative performance measurement (note that $\vartheta = -\pi$ has the same physical meaning as $\vartheta = \pi$, so the latter value is omitted from X_0).

Figure 4 shows the learning performance of the four ER algorithms (ER-Q-learning on separate samples and trajectories, and ER-SARSA on separate samples and trajectories). All the ER algorithms reliably lead to a good performance after collecting samples for only 60 s. The ER-Q-learning variants are better than the ER-SARSA variants by slight, but statistically significant margins.

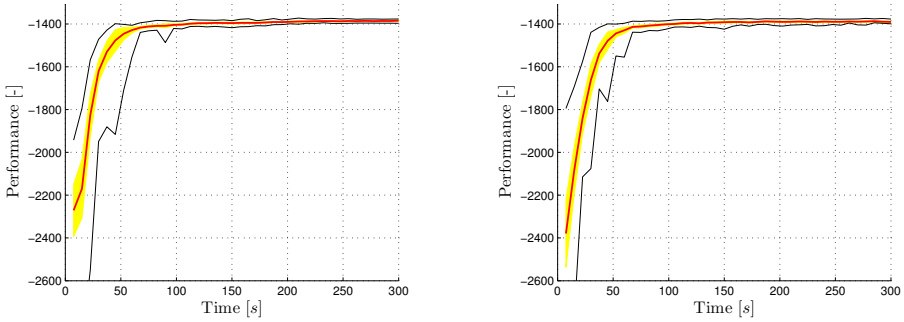
Note that in Figure 4, as well as in similar figures in the sequel, the time axis only measures the time spent *interacting* with the system. The time spent executing the underlying RL algorithm on the samples collected (11 of Algorithm 1), in-between trajectories, is not included. Initially, when only a few samples are available, this latter execution time is short, but it grows as more samples are collected, and eventually it may exceed the length of a sampling interval. This is not a problem in our experiments because updates are only performed in-between learning trials, when the system does not have to be controlled. When updates must be performed while controlling the system, steps must be taken to ensure that the real-time constraints are satisfied – e.g., by performing only a fixed number of updates, as discussed in Section III-A.

For comparison purposes, classical Q-learning and SARSA (without ER) are applied to the same problem. The (fixed) learning rate is $\alpha = 0.1$, and an eligibility trace with $\lambda = 0.9$ is used to speed up learning. These values have been hand-tuned to give the best performance. Trajectories are generated in the same way as for the ER algorithms. The results for SARSA are shown in Figure 5. The results for Q-learning are not shown because Q-learning diverges – even though ER-Q-learning has worked well. This may be because ER-Q-learning holds the policy constant along each trajectory, which increases learning stability (see Section III).[5] SARSA achieves its maximum performance in approximately 800 s, a much longer interval than required by the ER algorithms in Figure 4. This illustrates the poor data efficiency of SARSA. Additionally, SARSA has a worse final performance than the ER algorithms.

Since ER RL can be seen as bridging incremental and batch RL algorithms, an interesting question is how it compares with a batch algorithm; we select least-squares policy iteration (LSPI) [17] to perform this comparison. At every iteration, LSPI uses a batch of samples to compute the Q-function under the current policy (policy evaluation), and then finds an improved, greedy policy (4) in this Q-function. Exploiting the

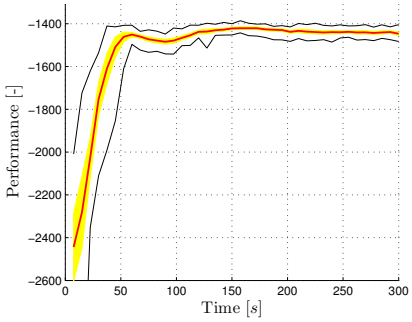
[4]Exponential decay does not asymptotically lead to infinite exploration, which is in principle required by Q-learning and SARSA (see Section II-B). Nevertheless, for an experiment having a finite duration, ε_{d} can be chosen large enough to provide any desired amount of exploration.

[5]Thus, in future work it may also be interesting to perform an experiment where (non-ER) Q-learning updates the policy only at the end of each trajectory, like the ER variant.

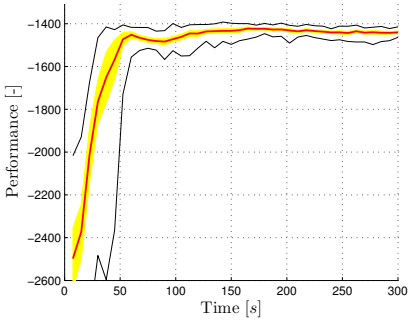


(a) ER-Q-learning on separate samples.

(b) ER-Q-learning on trajectories.



(c) ER-SARSA on separate samples.



(d) ER-SARSA on trajectories.

Fig. 4. Performance of ER-Q-learning and ER-SARSA in simulation. The thick line is the mean, thin lines show the minimal and maximal performance obtained over 20 independent runs, and the shaded region shows the 95% confidence interval of the mean.

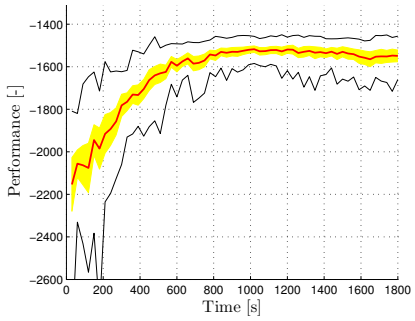


Fig. 5. Performance of classical SARSA in simulation.

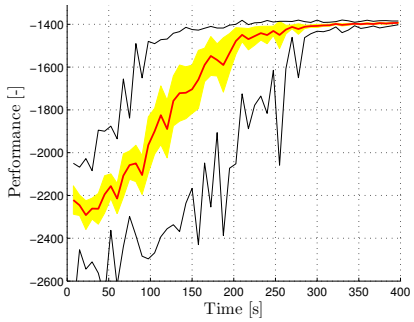


Fig. 6. Performance of LSPI in simulation.

fact that different batches of samples can be used at different iterations, we interleave LSPI with episodes of online sample collection, in a similar way to how the ER algorithms work. In particular, a T-steps long trajectory is generated online, using the current policy combined with exploration, and added to the database. Then, one iteration of LSPI (policy evaluation and improvement) is performed using the data collected thus far, and the whole cycle repeats.

LSPI is applied to the inverted pendulum using the same Q-function approximator as in the other algorithms. As shown in Figure 6, LSPI reaches a near-optimal performance after collecting samples for approximately 300 s, longer than required by the ER algorithms. Furthermore, note that LSPI must solve a linear system in n variables at each iteration; hence, its computational complexity is at least quadratic in n, whereas the complexity of the ER algorithms is linear in n, see (12).

To illustrate the quality of the ER solution, Figure 7 shows a trajectory starting in the stable equilibrium (pointing down) and controlled with the policy found by ER-Q-learning on separate samples. The pendulum is successfully swung up and stabilized in around 1 s. Note that the control action chatters; this is necessary because the pendulum must be stabilized around its unstable equilibrium using only discrete actions.

3) Results for the real inverted pendulum: Next, the ER algorithms are applied to the real inverted pendulum system. The parameters α, N, and the exploration schedule are the same as in simulation. While the initial states for every trajectory were uniformly random in simulation, obtaining such initial states is nontrivial for the real system. Instead, to obtain random (but not uniformly distributed) initial states, $u_1 = \pm u_{max}$ is applied for a random interval of time. The choice of the sign is also random. Then, $u_2 = -u_1$ is applied for another random interval. To evaluate the performance, a heuristic controller (approximately) brings the system to every state in the set $X_0 = \{[-\pi/2, 0]^\mathrm{T}, [0, -2\pi]^\mathrm{T}, [\pi/2, 0]^\mathrm{T}, [0, 2\pi]^\mathrm{T}\}$, the system is

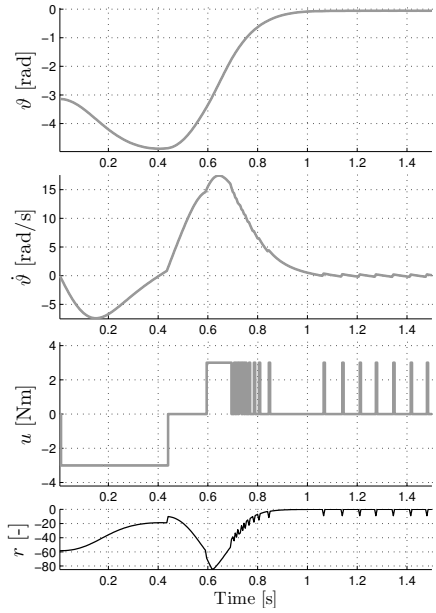


Fig. 7. A trajectory of the inverted pendulum from $x_0 = [-\pi, 0]^{\mathrm{T}}$, controlled by a policy found with ER-Q-learning. An exact multiple of 2π is added to the initial angle to account for full rotations made during the trajectory.

controlled with the current policy from every such state, and finally, the returns obtained are averaged across the set X_0.

The learning performance of the ER algorithms on the real system is shown in Figure 8.[6] All ER algorithms effectively learn to swing up and stabilize the real pendulum. Note that the experimental results of Figure 8 cannot be directly compared to the simulation results of Figure 4, since the set X_0 is different from the one used to evaluate the simulation results. Moreover, the dynamics of the real system are close, but not identical to the simulation model (for instance, Coulomb friction and stiction were not simulated). All the ER algorithms obtain a similar final performance, but ER-Q-learning on trajectories learns significantly slower than the other three algorithms – unlike in simulation where it performed equally well. This performance degradation might be due to sensitivity to the noise present in the real system, or to the nonuniform distribution of the starting states. Compared to the simulation results of Figure 4, the variance in the performance of all the ER algorithms is larger, due to noise during learning and performance evaluation. Also, in order to converge, ER-Q-learning on samples and both variants of ER-SARSA require more samples than they needed in simulation: they collect samples for 100 s, compared to 60 s in simulation.

[6] See also http://www.youtube.com/watch?v=b1c0N_Fs9wc for a movie showing how ER-Q-learning on trajectories learns to swing up the pendulum.

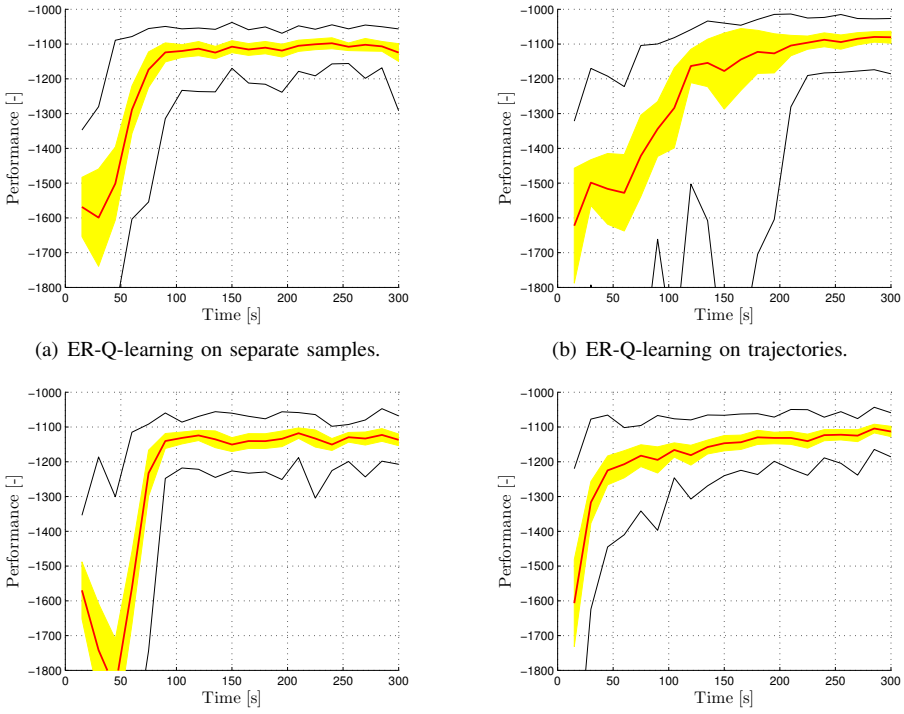
B. *Two-link robotic manipulator*

In this section, the ER algorithms are used to stabilize a simulated two-link manipulator operating in a horizontal plane. This illustrates how the ER algorithms scale up to a higher-dimensional problem than the inverted pendulum: the robotic manipulator has four state variables and two action variables.

1) Robotic manipulator problem: The robotic manipulator, shown in Figure 9, is described by the model:

$$M(\vartheta)\ddot{\vartheta} + C(\vartheta, \dot{\vartheta})\dot{\vartheta} = \tau \tag{13}$$

where $\vartheta = [\vartheta_1, \vartheta_2]^{\mathrm{T}}$, $\tau = [\tau_1, \tau_2]^{\mathrm{T}}$. The state vector contains the angles and angular velocities of the two links: $x =$



(a) ER-Q-learning on separate samples.

(b) ER-Q-learning on trajectories.

(c) ER-SARSA on separate samples.

(d) ER-SARSA on trajectories.

Fig. 8. Performance of ER-Q-learning and ER-SARSA for the real pendulum.

$[\vartheta_1, \dot{\vartheta}_1, \vartheta_2, \dot{\vartheta}_2]^{\mathrm{T}}$, and the control vector is $u = \tau$. The angles "wrap around" in the interval $[-\pi, \pi)$ rad, and the velocities (measured in rad/s) and torques (measured in Nm) are bounded as shown in Table I, which also collects the meaning and values of other variables in the system. The sampling time is $T_{\mathrm{s}} = 0.05$ s. For the matrices M and C, see Sec. 4.5.2 of [6].

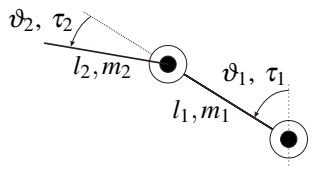


Fig. 9. A schematic representation of the robotic manipulator.

TABLE I
PARAMETERS OF THE TWO-LINK MANIPULATOR

Symbol	Domain or value. Units	Meaning
g	9.81 m/s^2	gravitational acceleration
ϑ_1; ϑ_2	$[-\pi, \pi)$; $[-\pi, \pi)$ rad	link angles
$\dot{\vartheta}_1$; $\dot{\vartheta}_2$	$[-2\pi, 2\pi]$; $[-2\pi, 2\pi]$ rad/s	link angular velocities
τ_1; τ_2	$[-1.5, 1.5]$; $[-1, 1]$ Nm	motor torques
l_1; l_2	0.4; 0.4 m	link lengths
m_1; m_2	1.25; 0.8 kg	link masses
I_1; I_2	0.066; 0.043 kg m^2	link inertias
c_1; c_2	0.2; 0.2 m	centers of mass
b_1; b_2	0.08; 0.02 kg/s	dampings in the joints

The control goal is stabilizing the system around the equilibrium $\vartheta = \dot{\vartheta} = 0$ and is expressed by the reward function:

$$\overline{\rho}(x,u) = -x^{\mathrm{T}} Q_{\mathrm{rew}} x, \text{ with } Q_{\mathrm{rew}} = \mathrm{diag}[1, 0.05, 1, 0.05] \quad (14)$$

The discount factor is $\gamma = 0.98$.

The Q-function approximator is similar to the one used for the inverted pendulum of Section IV-A. A number of 5^4 identically-shaped RBFs are defined, distributed on an equidistant grid with 5 points along each axis of the state space. The action space is discretized in the set $U_{\mathrm{d}} = \{-1.5, 0, 1.5\} \times \{-1, 0, 1\}$. This yields a total of $5^4 \cdot 3^2 = 5625$ state-action BFs. These BFs are chosen so that an accurate enough approximator is obtained, without leading to excessive computational costs. As in Section IV-A, the learning performance is measured by estimating the average return over a set X_0 of representative initial states. For the robotic manipulator, this set contains a regular grid of initial angles, while the initial velocities are always zero:

$$X_0 = \{-\pi, -2\pi/3, -\pi/3, 0, \pi/3, 2\pi/3, \pi\} \times \{0\} \\ \times \{-\pi, -2\pi/3, -\pi/3, 0, \pi/3, 2\pi/3, \pi\} \times \{0\}$$

2) Results: To apply the ER algorithms, trajectories started in uniformly random initial states and having a length of 5 s are used. Each such trajectory contains $T = 100$ samples. After each trajectory, the samples are presented to the underlying RL algorithms $N = 10$ times, using a fixed learning rate $\alpha = 0.3$. The exploration probability is initialized at 1, and decays exponentially with a rate of 0.9886 at the end of each trajectory. Figure 10 shows the learning performance of the four ER algorithms. The ER-SARSA algorithms converge to a good performance more reliably than the ER-Q-learning variants. The performance of the ER-Q-learning variants reaches a good value fast, but then degrades significantly before recovering again. This leads to an undesirable "dip" in the performance; the cause of this phenomenon is unclear. The algorithms that replay entire trajectories learn faster than those that replay separate samples. The ER algorithms reach a good performance after collecting samples for 400 to 800 s, while they required 60 s in the inverted pendulum problem. However, the sampling time for the robotic manipulator is 10 times larger than for the inverted pendulum (0.05 rather than 0.005 s), and so 10 times fewer samples are obtained in a given interval of time. This means that roughly as many samples are required as for the inverted pendulum, despite the larger number of BFs (5625, rather than 363). Overall, the ER algorithms scale up well to the higher-dimensional robotic manipulator problem.

To illustrate the quality of the ER solutions, Figure 11 shows a trajectory controlled with the policy found by ER-Q-learning on trajectories. In order to quickly reach the origin, the controller pushes the two links in opposite directions and then stabilizes them.

Note that we have also attempted to solve the robotic manipulator problem with the original Q-learning and SARSA algorithms, but they failed to reach a good solution.

C. Robotic soccer goalkeeper

In this section, ER RL is applied to control a real-life robotic goalkeeper, illustrating the ability of ER RL to perform well in a realistic application.

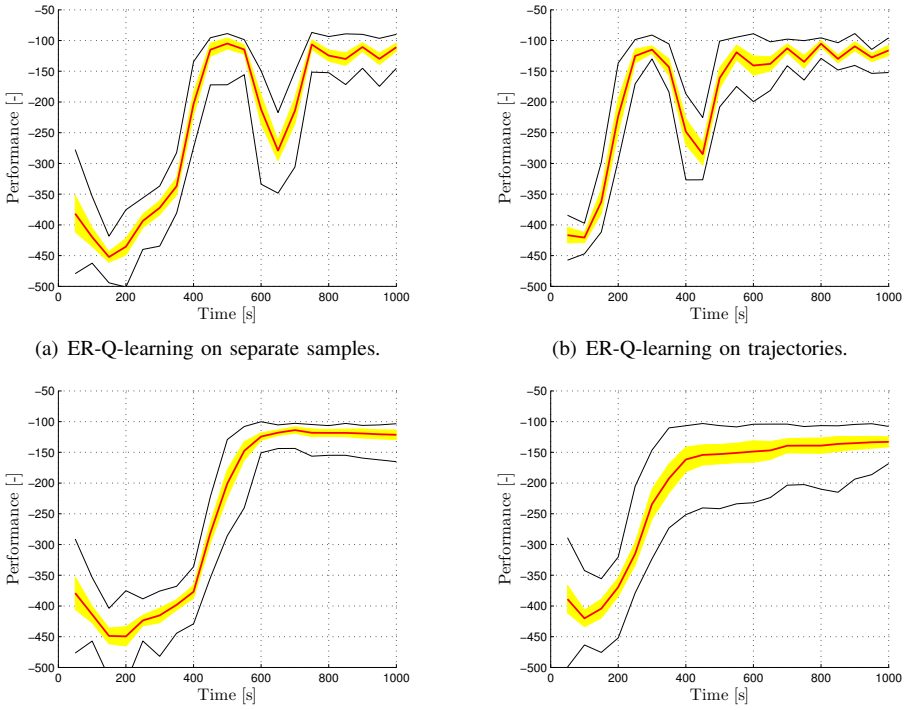
1) Robotic goalkeeping problem: Robotic soccer is a popular benchmark problem for RL [22], [30]. Robotic soccer typically involves multiple robots that must navigate, catch and shoot the ball, and, above all, cooperate with their teammates while competing with the opposing team. This is an extremely difficult problem that cannot be fully solved using only currently available RL algorithms. Therefore, obtaining meaningful benchmark results for RL in robotic soccer is not an easy task.

To address this issue, we have developed a simpler, but still challenging sub-problem of robotic soccer. In this problem, a fixed robotic goalkeeper must catch balls shot towards the goal. The goalkeeper is a robotic arm that tracks the ball using a camera, as shown in Figure 12. The dimensions of the goalkeeper are based on the small-size robotic soccer league. To be able to defend the entire goal, the main arm of the goalkeeper is 0.2 m long. The main goal of the camera arm is to point the camera towards the ball, so this arm is only 0.05 m long. The goalkeeper is designed to be light and stiff, so that it can robustly endure many RL trials.

To fully describe the goalkeeper system, 8 state variables are required: the angles and angular velocities of the two links; and the angle, angular velocity, distance, and linear velocity of the ball relative to the camera. The actions are the torques of the two motors that rotate the links:

$$x = [\vartheta_1, \dot{\vartheta}_1, \vartheta_2, \dot{\vartheta}_2, \vartheta_{\mathrm{b}}, \dot{\vartheta}_{\mathrm{b}}, d_{\mathrm{b}}, \dot{d}_{\mathrm{b}}]^{\mathrm{T}}, \quad u = [\tau_1, \tau_2]^{\mathrm{T}}$$

The high dimensionality of this problem, combined with the very low number of samples that can be collected, pose



(a) ER-Q-learning on separate samples.

(b) ER-Q-learning on trajectories.

(c) ER-SARSA on separate samples.

(d) ER-SARSA on trajectories.

Fig. 10. Performance of ER-Q-learning and ER-SARSA for the robotic manipulator. The thick line is the mean of 20 independent runs, thin lines show the minimal and maximal performance across these runs, and the shaded region shows the 95% confidence interval of the mean.

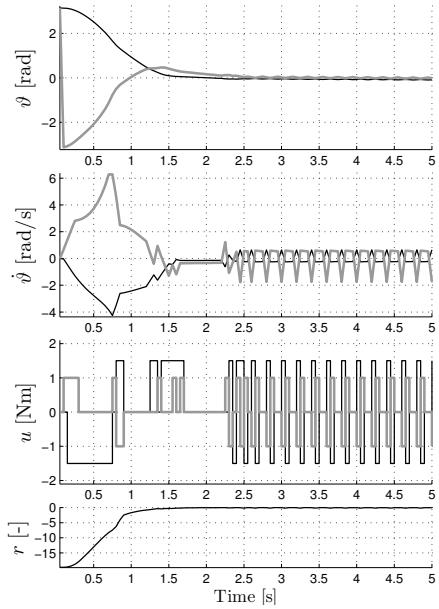


Fig. 11. A trajectory of the manipulator from $x_0 = [\pi, 0, \pi, 0]^{\mathrm{T}}$, controlled by a policy found with ER-Q-learning on trajectories. Black lines correspond to the first link, and gray lines to the second.

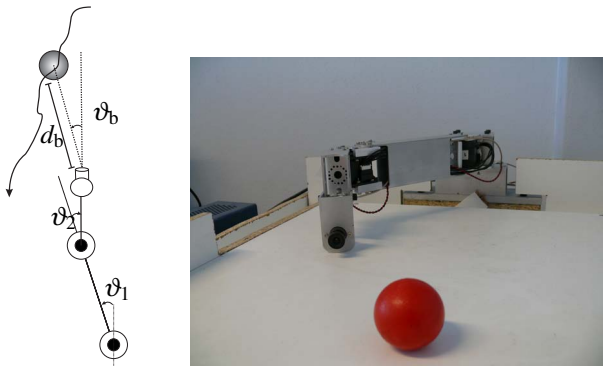


Fig. 12. A schematic representation of the robotic goalkeeper (left), and the real system (right).

great challenges to RL. Since a ball shot at the goal takes roughly one second, and using a camera frame rate of 25 frames per second, only 25 samples are generated during each shot. We identify each shot with a learning trial. For practical considerations, the goalkeeper is required to learn within at most 50 trials, so the total number of samples available is roughly 1250. Given the high dimensionality of the state-action space, this number of samples is clearly insufficient, even if they are replayed using ER. Therefore, the problem will have to be significantly simplified to make learning feasible.

The problem is simplified by eliminating some of the state and action variables. Since the ball will always be in front of the arm, it is intuitive to make the camera always point forward (when the ball has passed the arm, useful samples can no longer be collected). So, with a fast pre-designed controller that only relies on internal sensors (and not on the camera), τ_2 is controlled so that ϑ_2 follows $-\vartheta_1$. Therefore, the RL controller only needs to learn to control τ_1, and τ_2 is eliminated from its input variables. The settling time for this control loop is smaller than the camera sampling time, so from the point of view of the RL controller ϑ_2 is always equal to $-\vartheta_1$. So, ϑ_2 can be eliminated as well. Moreover, since the

image recognition signal is noisy, the estimates of $\dot{\vartheta}_\mathrm{b}, d_\mathrm{b}$ and $\dot{d}_\mathrm{b}$ are limited in their use, and they are also not presented to the controller. This essentially reduces the goalkeeping problem to a tracking problem. To catch the ball, the angle ϑ_b needs to be minimized. Since this tracking problem is approximately the same for every angle ϑ_1, the variable ϑ_1 can also be eliminated. The remaining state vector is denoted by $\tilde{x} = [\vartheta_\mathrm{b}, \dot{\vartheta}_1]^\mathrm{T}$, and the remaining action is $\tilde{u} = \tau_1$. The reward function chosen to express the goal of tracking and catching the ball is:

$$\overline{\rho}(\tilde{x}, \tilde{u}) = -\tilde{x}^\mathrm{T} Q_\mathrm{rew} \tilde{x}, \quad \text{with } Q_\mathrm{rew} = \mathrm{diag}[1, 0.01]$$

The discount factor is $\gamma = 0.8$. As in Sections IV-A and IV-B, the Q-function approximator uses state-dependent RBFs and discrete actions. Identically-shaped RBFs are distributed on an equidistant grid with 4 points along the domain of each variable ϑ_b and $\dot{\vartheta}_1$.

2) Results: To apply the ER algorithms, we use ball shot trials instead of fixed-length trajectories in the ER framework of Algorithm 1, see Section III. After each ball shot, the samples are reused $N = 10$ times, using a fixed learning rate $\alpha = 0.2$. Two additional improvements are made to speed up the learning, as described next. Many discretized-action problems, including the goalkeeper, have the following useful property: in optimal trajectories there exist long subsequences of states along which the optimal action does not change (for an example, see the pendulum swing-up trajectory of Figure 7). To exploit this property, we use the *dynamic exploration* strategy proposed by [35], which modifies any basic exploration strategy by introducing an inertia factor $\beta \in [0, 1]$. The inertia factor encodes a tendency to select u_k equal to u_{k-1} at every step k. We set $\beta = 0.5$, and combine dynamic exploration with soft-max exploration [32]. This increases the learning speed compared to ε-greedy exploration. We also use easy missions [28] to speed up learning. With easy missions, each learning trial is split in two phases. In the first phase, the process is led to a highly rewarding state by a heuristic controller. In the second phase, the RL controller takes over. The length of the first phase is reduced as learning progresses, so that eventually the RL controller leads the entire trial. For the first ball shot, the entire trajectory is guided by the heuristic controller. The length of the guided phase decreases with 0.08 s after each ball shot, so that the RL controller is fully leading the system after 25 shots.

With these improvements, after learning for 50 trials, the goalkeeper is able to reliably catch the ball. The performance is still suboptimal at this point, and learning for a longer time would improve the results further. Figure 13 shows the policy obtained by ER-SARSA on separate samples.[7]

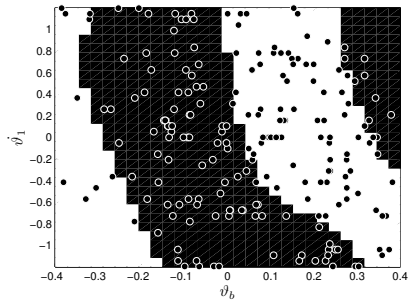


Fig. 13. Goalkeeper policy learned with ER-SARSA on separate samples. Black represents the maximum negative action, white maximum positive action, and the circles mark the locations of state samples obtained (only 10% of the samples are shown, to avoid cluttering the figure).

V. Conclusions and Future Work

This paper has investigated the practical performance of a class of experience replay reinforcement learning (ER RL) methods. ER increases the data efficiency of an underlying RL algorithm by repeatedly presenting the available data to this algorithm. We described a general framework that combines ER with any incremental RL technique, and instantiated this framework for approximate Q-learning and SARSA. ER-Q-learning and ER-SARSA performed well in an extensive evaluation on a range of real and simulated applications, involving an inverted pendulum, a robot arm, and a goalkeeper robot. Among other results, ER outperformed classical Q-learning and SARSA, as well as the batch LSPI algorithm.

A first opportunity for further research is to further speed up learning by replaying samples in backward temporal order, instead of temporal or random order; alternatively, the most promising samples could be replayed first using the so-called prioritized sweeping [21]. Another important question is how the database pruning mechanisms suggested in Section III-A would affect the performance. It would also be interesting to compare ER learning with other batch algorithms besides LSPI, for instance, with fitted Q-iteration employing a non-parametric approximator [11].

In order to stimulate real-world applications of RL, it is necessary to demonstrate that RL methods can be effectively used in the control of physical systems. The successful real-time learning control results presented in this paper are in our view an important step in this direction.

[7] See also http://www.youtube.com/watch?v=CIF2SBVY-J0 for a movie of the learning process.

References

[1] L. Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings 12th International Conference on Machine Learning (ICML-95)*, pages 30–37, Tahoe City, US, 9–12 July 1995.

[2] F. Bernardo, R. Agustí, J. Pérez-Romero, and O. Sallent. An application of reinforcement learning for efficient spectrum usage in next-generation mobile cellular networks. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 2010. In press.

[3] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 2. Athena Scientific, 3rd edition, 2007.

[4] D. P. Bertsekas and S. E. Shreve. *Stochastic Optimal Control: The Discrete Time Case*. Academic Press, 1978.

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

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

[7] L. Buşoniu, D. Ernst, B. De Schutter, and R. Babuška. Online least-squares policy iteration for reinforcement learning control. In *Proceedings 2010 American Control Conference (ACC-10)*, pages 486–491, Baltimore, US, 30 June – 2 July 2010.

[8] P. Cichosz. An analysis of experience replay in temporal difference learning. *Cybernetics and Systems*, 30:341–363, 1999.

[9] L. T. Dung, T. Komeda, and M. Takagi. Efficient experience reuse in non-markovian environments. In *Proceedings 2008 International Conference on Instrumentation, Control and Information Technology (SICE-08)*, pages 3327–3332, Tokyo, Japan, 20–22 August 2008.

[10] D. Ernst. *Near Optimal Closed-loop Control. Application to Electric Power Systems.* PhD thesis, University of Liège, Belgium, March 2003.

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

[12] D. Ernst, G.-B. Stan, J. Gonçalves, and L. Wehenkel. Clinical data based optimal STI strategies for HIV: A reinforcement learning approach. In *Proceedings 45th IEEE Conference on Decision & Control*, pages 667–672, San Diego, US, 13–15 December 2006.

[13] D. Gu and H. Hu. Integration of coordination architecture and behavior fuzzy learning in quadruped walking robots. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 37(4):670–681, 2007.

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

[15] L. Jouffe. Fuzzy inference system learning by reinforcement methods. *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, 28(3):338–355, 1998.

[16] S. Kalyanakrishnan and P. Stone. Batch reinforcement learning in a complex domain. In *Proceedings 6th International Conference on Autonomous Agents and Multi-Agent Systems*, pages 650–657, Honolulu, US, 14–18 May 2007.

[17] M. G. Lagoudakis and R. Parr. Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149, 2003.

[18] L. Li, M. L. Littman, and C. R. Mansley. Online exploration in least-squares policy iteration. In *Proceedings 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-09)*, volume 2, pages 733–739, Budapest, Hungary, 10–15 May 2009.

[19] L.-J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3–4):293–321, Aug. 1992. Special issue on reinforcement learning.

[20] F. S. Melo, S. P. Meyn, and M. I. Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings 25th International Conference on Machine Learning (ICML-08)*, pages 664–671, Helsinki, Finland., 5–9 July 2008.

[21] A. W. Moore and C. G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

[22] K. Park, Y. Kim, and J. Kim. Modular Q-learning based multi-agent cooperation for robot soccer. *Robotics and Autonomous Systems*, 35:109–122, 2001.

[23] T. J. Perkins and D. Precup. A convergent form of approximate policy iteration. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1595–1602. MIT Press, 2003.

[24] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21:682–697, 2008.

[25] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR166, Engineering Department, Cambridge University, UK, September 1994. Available at http://mi.eng.cam.ac.uk/reports/svr-ftp/rummery_tr166.ps.Z.

[26] S. Singh and R. Sutton. Reinforcement learning with replacing eligibility traces. *Machine Learning*, 22:123–158, 1996.

[27] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement learning with soft state aggregation. In G. Tesauro, D. S. Touretzky, and T. K. Leen, editors, *Advances in Neural Information Processing Systems 7*, pages 361–368. MIT Press, 1995.

[28] W. Smart. *Making Reinforcement Learning Work on Real Robots.* PhD thesis, Brown University, 2002.

[29] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings 17th International Conference on Machine Learning (ICML-00)*, pages 903–910, Stanford University, US, 29 June – 2 July 2000.

[30] P. Stone, R. Sutton, and G. Kuhlmann. Reinforcement learning for RoboCup soccer keepaway. *Adaptive Behavior*, 13:165, 2005.

[31] R. S. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Proceedings 7th International Conference on Machine Learning (ICML-90)*, pages 216–224, Austin, US, 21–23 June 1990.

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

[33] R. S. Sutton, Cs. Szepesvári, A. Geramifard, and M. H. Bowling. Dyna-style planning with linear function approximation and prioritized sweeping. In *Proceedings 24th Conference in Uncertainty in Artificial Intelligence (UAI-08)*, pages 528–536, Helsinki, Finland, 9–12 July 2008.

[34] Cs. Szepesvári and W. D. Smart. Interpolation-based Q-learning. In *Proceedings 21st International Conference on Machine Learning (ICML-04)*, pages 791–798, Bannf, Canada, 4–8 July 2004.

[35] J. Van Ast and R. Babuska. Dynamic exploration in Q(λ)-learning. In *Proceedings of International Joint Conference on Neural Networks*, pages 41–46, Vancouver, Canada, 16–21 July 2006.

[36] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

[37] P. Wawrzynski. Real-time reinforcement learning by sequential actor-critics and experience replay. *Neural Networks*, 22(10):1484–1497, 2009.

Sander Adam received his MSc degree (cum laude) in 2008 at the Delft Center for Systems and Control of Delft University of Technology, in the Netherlands. His research interest was focused on the practical applications of reinforcement learning. Currently, Sander is employed in the Large Corporates & Merchant Banking division of ABN AMRO Bank in the Netherlands.

Lucian Buşoniu received his PhD degree (cum laude) in 2009 from the Delft University of Technology, and his MSc degree (valedictorian) in 2003 from the Technical University of Cluj-Napoca, Romania. His current research interests include reinforcement learning and dynamic programming with function approximation, intelligent and learning techniques for control problems, and multi-agent learning.

Robert Babuška is a professor of Intelligent Control and Robotics at the Delft Center for Systems and Control, Delft University of Technology in the Netherlands. He received his PhD degree (cum laude) in Control in 1997 from the Delft University of Technology, and his MSc degree (with honors) in Electrical Engineering in 1990 from Czech Technical University, Prague. His research interests include fuzzy systems modeling and identification, data-driven construction and adaptation of neuro-fuzzy systems, model-based fuzzy control and reinforcement learning control. He is active in applying these techniques in robotics, mechatronics, and transportation systems.