DSMC Evaluation Stages: Fostering Robust and Safe Behavior in Deep Reinforcement Learning

Timo P. Gros(✉), Daniel Höller, Jörg Hoffmann, Michaela Klauck, Hendrik Meerkamp, and Verena Wolf

Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
{timopgros,hoeller,hoffmann,klauck,meerkamp,wolf}@cs.uni-saarland.de

Abstract. Neural networks (NN) are gaining importance in sequential decision-making. Deep reinforcement learning (DRL), in particular, is extremely successful in learning action policies in complex and dynamic environments. Despite this success however, DRL technology is not without its failures, especially in safety-critical applications: (i) the training objective maximizes *average* rewards, which may disregard rare but critical situations and hence lack local robustness; (ii) optimization objectives targeting safety typically yield degenerated reward structures which for DRL to work must be replaced with proxy objectives. Here we introduce methodology that can help to address both deficiencies. We incorporate *evaluation stages* (ES) into DRL, leveraging recent work on deep statistical model checking (DSMC) which verifies NN policies in MDPs. Our ES apply DSMC at regular intervals to determine state space regions with weak performance. We adapt the subsequent DRL training priorities based on the outcome, (i) focusing DRL on critical situations, and (ii) allowing to foster arbitrary objectives. We run case studies in Racetrack, an abstraction of autonomous driving that requires navigating a map without crashing into a wall. Our results show that DSMC-based ES can significantly improve both (i) and (ii).

1 Introduction

In recent years, neural networks (NN), especially deep neural networks, have accomplished major successes across many computer science domains, like image classification [25], natural language processing [21], and game-playing [41]. The latter was especially accomplished by combining reinforcement learning (RL) and deep neural networks, so called *deep reinforcement learning* (DRL). DRL was used successfully for sequential decision-making, e.g., mastering Atari games [28,29], playing the games Go and Chess [40–42], or solving the Rubik's cube [1],

Authors are listed alphabetically. This work was partially supported by the German Research Foundation (DFG) under grant No. 389792660, as part of TRR 248, see https://perspicuous-computing.science, and by the European Regional Development Fund (ERDF).

The original version of this chapter was revised: an error in the algorithm on page 206 were corrected. The correction to this chapter is available at https://doi.org/10.1007/978-3-030-85172-9_25

and is beginning to be used in real-world (motivated) examples, such as vehicle routing [30], robotics [17], and autonomous driving [36].

Despite this success however, DRL technology is not without fail, especially in safety-critical applications. While neural network action policies achieve good performance in many sequential decision-making processes, that performance pertains to *average* rewards as optimized by DRL training. That objective however may average out poor local behavior, and thus disregard rare but critical situations (e.g. a child running in front of a car). In other words, we do not get system-level guarantees, even in the ideal case where the learned policy is near-optimal with respect to its training objective. We refer to this deficiency as a lack of *local robustness*. Dedicated exploration strategies have been developed to ensure inclusion of rare experiences during training [11,12]. These focus on reducing the variance of the accumulated reward, e.g. by importance sampling, but they are not flexible enough to enforce desirable behavior robustly across the whole state space.

This problem is exacerbated by the fact that optimization objectives specifically targeting safety typically yield degenerated reward structures. This is true in particular for the natural objective to maximize the probability of reaching a goal condition – without getting stuck in an unsafe (terminal) state. That objective yields reward 1 in goal states and 0 elsewhere, an extremely sparse reward structure not suited for (D)RL training in large state spaces (a widely known fact, see e.g. [3,18,24,35,38]). Hence, for (D)RL training to be able to identify a useful policy, proxy objectives are used, such as discounted cumulative reward giving positive feedback for goal states and (highly) negative feedback for unsafe states.[1]

In summary, two deficiencies of current DRL methods in safety-critical systems are that (i) training for average reward lacks local robustness, and (ii) safety objectives like goal probability cannot be used for effective training. Let us illustrate these points in an example, taken from the Racetrack benchmark which we will also use in our case studies later on. Racetrack is a commonly used benchmark for Markov decision process (MDP) solution algorithms in AI [6,8,33,44]. The challenge is to navigate to a goal line on a discrete map without crashing into a wall, where actions accelerate/decelerate the car. Racetrack is thus a simple (but highly extensible [5]) abstraction of autonomous driving. Consider Fig. 1, which measures the performance of a NN policy trained with DRL, using a discounted reward structure with $+100$ reward for goal states and -50 reward for crashes.

Figure 1 (a) evaluates policy performance according to the reward structure it is trained on; whereas Fig. 1 (b) evaluates goal probability, which is the objective we ideally want to optimize. Both heat maps visualize performance when

[1] One can combine such a proxy with the goal probability objective, though multiple objectives are difficult to achieve with a one-dimensional reward signal and standard backpropagation algorithms for neural nets [26]; anyway, training objective vs. ideal objective are still not identical here. Reward shaping is an alternative option that can in principle preserve the optimal policy [31], but this is not always possible, and manual work is needed for individual learning tasks (substantial work sometimes, see e.g. [46]).

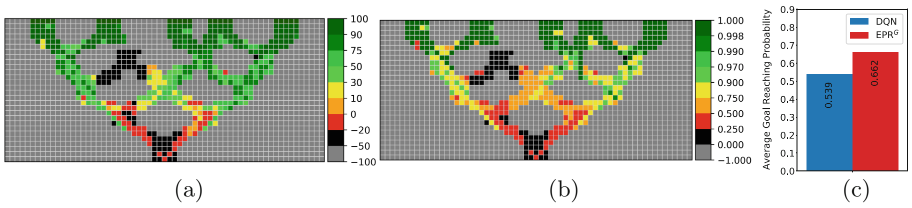


Fig. 1. Example performance measures of a DRL policy on a Racetrack example map. (Color figure online)

starting the policy from each map cell. We clearly see deficiency (i) from the high variance in colors, in particular black and red areas with (very) low expected reward (a) /goal probability (b). Regarding deficiency (ii), while expected reward correlates with goal probability, crashes are "more tolerable" in the reward structure than for goal probability (if we set high negative rewards for crashes then the policy learns to drive in circles). This is difficult to see in the heat maps as the reward scale in (a) cannot be directly compared to the probability scale in (b). Figure 1 (c) hence complements this picture by average goal probability in the critical areas of the map, as achieved by the standard DRL method *deep Q-network*, vs. EPR^G which is one of the new methods we introduce here. EPR^G takes goal probability into account directly, which clearly pays off.

We address deficiencies (i) and (ii) through incorporating *evaluation stages (ES)* into DRL, conducted at regular intervals during training (i.e., periodically after a given number of training episodes) to determine state space regions with weak performance. The "performance" evaluation here is flexible, and can be done either (i) with respect to the training objective, or (ii) with respect to the true objective (for example: goal probability in EPR^G above) in case a proxy objective is used for training.

To design such flexible ES, we leverage recent work on *deep statistical model checking* (DSMC) [15], an approach to explicitly examine properties of NN action policies. The approach assumes that the NN policy resolves the nondeterminism in an MDP, resulting in a Markov chain which is analyzed by statistical model checking [10]. This provides flexible methodology for evaluating policy performance starting from individual states.

The target of an evaluation stage being to identify "weak regions", the question arises which individual states to apply DSMC to. Our answer to this question, at present, is based on the assumption that the possible initial states for the problem at hand (the states from which policy execution may start) can be partitioned into a feasibly small set of state-space regions. In Racetrack, regions are identified by the location of the car. The approach we propose is to sample a single representative state s from each region, and evaluate s through DSMC.[2]

[2] The benefit of our proposed ES thus hinges, in particular, on how meaningful these representative states are for policy performance. While this is a limitation, partitioning by physical location like in Racetrack could be a canonical candidate in many scenarios.

Upon termination of an ES, we adapt the subsequent DRL training priorities based on the outcome. Specifically, we introduce two alternative methods, of which one adapts the probabilities with which new training experiences are generated, and the other adapts the probabilities with which the accumulated training experiences are taken into consideration within individual learning steps. Overall, this approach results in an iterative feedback loop between DRL training and DSMC model checking. It addresses (i) through focusing the DRL on critical situations, and addresses (ii) as DSMC can evaluate arbitrary temporal properties.

We implement this approach on top of deep Q-learning [6,29], and we run experiments in case studies from Racetrack. The results show that DSMC-based ES can indeed (i) make policy reward more robust across complex maps, and (ii) improve goal probability when using a discounted-reward proxy for DRL training.

In summary, our contributions are as follows:

– We introduce evaluation stages as an idea to improve local robustness and goal-probability training in DRL.
– We design and implement two variants of this approach, adapting a state-of-the-art DRL algorithm.
– We evaluate the approach in Racetrack and show that it can indeed have beneficial effects regarding deficiencies (i) and (ii).

Related Work: Recent work of Hasenbeig et al. [20] proposes a method to include a property encoded as an LTL formula and to synthesize policies that maximize the probability of that LTL property. However, while this allows to specify complex tasks, it addresses neither of our two deficiencies.

Further, our work relates to the area of *safe reinforcement learning*. Several works investigate the usage of *shields* [2,4,22] or *permissive schedulers* [23] to restrict the agent from entering unsafe states, even during training. However, these approaches can only be applied if a shield/permissive scheduler was computed beforehand, which is a model-based task. In contrast, our approach is model-free; it does not need to compute a shield or permissive scheduler beforehand, and does not restrict the action (and thus also state-) space. Instead, the task is learned entirely through self-play and Monte Carlo-based evaluation runs. Moreover, our approach is also applicable in more general scenarios, when there are not just safe and unsafe states but more fine-grained state distinctions.

2 Background

We briefly introduce the necessary background on Markov decision processes, deep Q-learning, and deep statistical model checking.

2.1 Markov Decision Processes

The underlying model of both DSMC and DRL is that of a (state-discrete) Markov decision process in discrete time. Let $\mathcal{D}(S)$ denote the set of probability distributions over S for any non-empty set S.

Definition 1 (Markov Decision Process). *A* Markov decision process *(MDP) is a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mu \rangle$ consisting of a finite set of* states $\mathcal{S}$*, a finite set of* actions $\mathcal{A}$*, a partial* transition probability function $\mathcal{T} \colon \mathcal{S} \times \mathcal{A} \rightharpoonup \mathcal{D}(\mathcal{S})$*, and an* initial distribution $\mu \in \mathcal{D}(\mathcal{S})$*. We say that an action $a \in \mathcal{A}$ is* applicable *in state $s \in \mathcal{S}$ if $\mathcal{T}(s, a)$ is defined. We denote by $\mathcal{A}(s) \subseteq \mathcal{A}$ the set of actions applicable in s.*

MDPs are typically associated with a *reward* structure r, specifying numerical rewards that are obtained when following a transition, i.e., $r : \mathcal{S} \times A \times \mathcal{S} \to \mathbb{R}$. In the following, we call the support of μ the set of initial states I, i.e., $I = \{s \in \mathcal{S} \mid \mu(s) > 0\}$.

Usually, an MDP's behavior is considered jointly with an entity resolving the otherwise non-deterministic choices in a state. Given a state, a so-called *action policy* (or scheduler, or adversary) determines which of the applicable actions to apply.

Definition 2 (Action Policy). *A (history-independent)* action policy *is a function $\pi \colon \mathcal{S} \times \mathcal{A} \to [0, 1]$ such that $\pi(s, \cdot)$ is a probability distribution on $\mathcal{A}$ and, for all $s \in \mathcal{S}$, $\pi(s, a) > 0$ implies that $a \in \mathcal{A}(s)$.*

We remark that history-independent action policies are often also called *memoryless* because their decisions depend only on the given state and not on the history of formerly visited states. We call an action policy *deterministic* if in each state s, π selects an action with probability one. We then simply write $\pi(s)$ for the corresponding action.

In the sequel, for a given MDP $\mathcal{M}$ and action policy π, we will write $S_0, S_1, S_2, \ldots$ for the states visited at times $t = 0, 1, 2, \ldots$. Let A_t be the action selected by policy π in state S_t and $R_{t+1} = r(S_t, A_t, S_{t+1})$ the reward obtained when transitioning from S_t to S_{t+1} with action A_t. Note that – as we are dealing with finite-state MDPs – the probability measure associated with these random variables is well defined and $\{S_t\}_{t \in \mathbb{N}_0}$ is a Markov chain with state space $\mathcal{S}$ induced by policy π. For further details we refer to Puterman [34].

The induced Markov chain can be analyzed using statistical model checking [39,47]. For statistical model checking of MDPs, different approaches have been proposed to handle nondeterminism [7,10].

2.2 Deep Q-learning

In the following, let

$$G_t = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{1}$$

denote the discounted, accumulated reward, also called *return*, from time t on, where $\gamma \in [0, 1]$ is a discount factor, and T is the final time step [44]. The discount factor determines the importance between short and long term rewards; if $\gamma = 0$, the discounted return will be equal to the reward accumulated in one step only, if $\gamma = 1$ all future rewards will be worth the same, and if $\gamma \in (0, 1)$ the long term rewards will be less important than the short term ones.

Q-learning is a well known algorithm to approximate action policies that maximize said accumulated reward [6]. For a fixed policy π, the so-called *action-value* or *q-value* $q_\pi(s, a)$ at time t is defined as the expected return G_t that is achieved by taking an action $a \in A(s)$ in state s and following the policy π afterwards, i.e.,

$$q_\pi(s,a) = \mathbb{E}_\pi \left[G_t \middle| S_t = s, A_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right]. \tag{2}$$

Policy π is optimal, if it maximizes the expected return. We write $q_*(s, a)$ for the corresponding *optimal action-value*. Intuitively, the optimal action-value $q_*(s, a)$ is equal to the expected sum of the reward that we receive when taking action a from state s, and the (discounted) highest optimal action-value that we receive afterwards. For optimal π, the Bellman optimality equation [44] gives

$$q_*(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma \cdot \max_{a'} q_* \left(S_{t+1}, a'\right) \middle| S_t = s, A_t = a\right]. \tag{3}$$

Vice versa, one can evidently obtain the optimal policy if the optimal action values are known by selecting $\pi(s) = \operatorname{argmax}_{a \in A(s)} q_*(s, a)$.

By estimating the optimal q-values, one can obtain (an approximation of) an optimal policy. During tabular *Q-learning*, the action values are approximated separately for each state-action pair [6]. In the case of large state spaces, *deep Q-learning* can be used to replace the Q-table by a *neural network* (NN) as a function approximator [29]. NNs can learn low-dimensional feature representations and express complex non-linear relationships. Deep reinforcement learning is based on training deep neural networks to approximate optimal policies. Here, we consider a neural network with weights θ estimating the Q-value function as a *deep Q-network (DQN)* [28]. We denote this Q-value approximation by $Q_\theta(s, a)$ and optimize the network w.r.t. the target

$$y_\theta(s,a) = \mathbb{E}_\theta \left[R_{t+1} + \gamma \cdot \max_{a'} Q_\theta(S_{t+1}, a') \mid S_t = s, A_t = a\right], \tag{4}$$

where the expectation is taken over trajectories induced by the policy represented by the parameters θ. The corresponding loss function in iteration i of the learning process is

$$L(\theta_i) = \mathbb{E}_{\theta_i} \left[\left(y_{\theta'}(S_t, A_t) - Q_{\theta_i}(S_t, A_t)\right)^2\right]. \tag{5}$$

Here, the so-called *fixed target* means that in Eq. (5) θ' does not depend on the current iteration's weights of the (so called *local*) neural network θ_i but on weights that were stored in earlier iterations (so called *target* network), to avoid an unstable training procedure [29]. We approximate $\nabla L(\theta_i)$ and optimize the loss function by stochastic gradient descent.

In contrast to Mnih et al. [29], we do not update the target network after a fixed number of learning stochastic gradient descend update steps, but perform a soft update instead, i.e., whenever we update the local network in iteration i, the weights of the target network are given by $\theta' = (1-\tau)\cdot\theta_i + \tau\cdot\theta'$ with $\tau \in (0,1)$ [13,43].

Stochastic gradient descent assumes independent and identically distributed samples. However, when directly learning from self-play, this assumption is disrupted as the next state depends on the current decision. To mitigate this problem, we do not directly learn from observations, but store them in an *experience replay buffer* [29]. Whenever a learning step is performed, we uniformly sample from this replay buffer to consider (approximately) uncorrelated tuples. Thus, the loss is given by

$$L(\theta_i) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma\cdot\max_{a'} Q_{\theta'}(s',a') - Q_{\theta_i}(s,a)\right)^2\right]. \tag{6}$$

We generate our experience tuples by exploring the state space epsilon-greedily, i.e., during the Monte Carlo simulation we follow the policy that is implied by the current network weights with a chance of $(1-\epsilon)$ and otherwise choose a random action. We start with a high exploration coefficient $\epsilon = \epsilon_{start}$ and exponentially decay it, i.e., for every iteration set $\epsilon = \epsilon \cdot \epsilon_{decay}$ with $\epsilon_{decay} < 1$, until a certain threshold ϵ_{end} is met. Afterwards, we constantly use $\epsilon = \epsilon_{end}$. Common termination criteria for the learning process are fixing the number of episodes or using a threshold on the expected return achieved by the current policy. The overall algorithm is displayed in Sect. 3 (Algorithm 1), together with the extensions and changes we will introduce.

A common improvement to the DQN algorithm sketched above, which we will also consider in this paper, is the so-called *prioritized replay buffer* [37]. Not all samples are equally useful to improve the policy. In particular, those samples with a relatively small individual loss do not contribute to the learning process as much as those with a high loss. Thus, the idea of prioritized experience replay is to sample from the aforementioned replay buffer with a probability that reflects the loss. Specifically, the priority δ of a sample (s,a,s',r) in iteration i is given by

$$\delta = \left(\left(Q_{\theta'}(s,a) - \left(r + \gamma\cdot\max_{a'} Q_{\theta_i}(s',a')\right)\right) + \epsilon_p\right)^{\alpha}, \tag{7}$$

where ϵ_p is a hyperparameter to ensure that all samples have non-zero probability, and α is used to control the amount of prioritization. $\alpha = 0$ means that there is no prioritization, $\alpha = 1$ means full prioritization, $\alpha \in (0,1)$ defines a balance. In Eq. (6), instead of sampling uniformly, the probability at which a sample is picked from the buffer is then proportional to its priority, i.e. we divide the samples' priority by the sum of all priorities. In the following, we will abbreviate DQN with such prioritized experience replay as *DQNPR*.

2.3 Deep Statistical Model Checking

Deep Statistical Model Checking. [15] is a method to analyze a NN-represented policy π taking action decisions (resolving the nondeterminism) in an MDP $\mathcal{M}$.

Namely, the induced Markov chain $\mathcal{C}$ is examined by statistical model checking. Given an MDP $\mathcal{M}$, DSMC assumes that the policy π has been trained based on $\mathcal{M}$ completely prior to the analysis without influencing the training process at all. This approach is promising in terms of scalability as the analysis of $\mathcal{C}$ merely requires to evaluate the NN on input states: there is no need for other deeper and more complex NN analyses. Gros et al. [15] implemented this approach for the statistical model checker MODES [10] in the MODEST TOOLSET [19].

3 RL with Evaluation Stages

We now introduce our approach of RL with evaluation stages, addressing the DRL deficiencies discussed in the introduction: (i) training for average reward lacks local robustness; (ii) safety objectives like goal probability cannot be used for effective training. We next discuss a basic design decision, then describe our two alternative methods, and then specify how they are realized on top of deep Q-learning.

3.1 Initial State Partitioning and Notations

Recall that I denotes the initial states of the MDP, i.e., the support of the initial distribution μ. As already mentioned, an important premise of our work is that I can be partitioned into a manageable number of *regions*. We denote that partition by $\mathbb{P} = \{J_1, J_2, \ldots, J_k\}$ where the regions are non-empty $J_i \neq \emptyset$, cover the set of all initial states $\bigcup_{i \in 1,2,\ldots,k} J_i = I$, and are disjoint $J_i \cap J_j = \emptyset$ for $i \neq j$. During the evaluation stages, we consider one representative $s_i \in J_i$ from each region. The underlying assumption is that the representatives are sufficiently meaningful to identify important deficiencies in policy behavior.[3]

The evaluation stages may consider arbitrary optimization objectives in principle, and use arbitrary methods to measure the objective values of the states s_i. Here we compute E using DSMC, measuring expected reward or goal probability. We denote the outcome of evaluation as an *evaluation function*, a function $E : \mathbb{P} \to [0, 1]$ mapping each region J_i to the evaluation value of its representative state s_i. For optimization objectives that are not probabilities, we assume here a normalization step into the interval $[0, 1]$, with 0 being the worst value and 1 the best. In particular, for expected rewards, the natural method we use in our experiments is to set $E(r^{\min}) = 0$ and $E(r^{\max}) = 1$ and interpolate linearly in between.

We also use the representative states to define an initial probability distribution over the regions J_i:

$$\beta(P_i) = \mu(s_i) / \sum_{j=1}^{k} \mu(s_j) \tag{8}$$

[3] In our Racetrack case studies, we use the map cells as the basis of $\mathbb{P}$ – i.e., states sharing the same physical location. We believe that this partitioning method may work for many application scenarios involving physical space. Alternatively, one may, for example, partition state-variable ranges into intervals.

3.2 Evaluation-Based Initial Distribution (EID)

Given the initial distribution μ of the MDP, with the insights gained through the DSMC evaluation stages, we can adapt the initial distribution to guide the training process after an evaluation stage. Recall that β is the initial distribution of a region in the original MDP. The probability to start in a region J_i for the EID method is then given by

$$p(J_i) = \frac{(1 - E(J_i)) \cdot \beta(J_i)}{\sum\limits_{j} (1 - E(J_j)) \cdot \beta(J_j)}, \tag{9}$$

i.e., we shift the initial distribution for the regions such that we start with a higher probability in areas with low quality and vice versa. Once region J_i is selected, we uniformly sample a starting state from J_i.

The idea of EID is that by generating experiences from regions with poor behavior, we improve the robustness of the policy as the NN will learn to select the most appropriate actions in these regions.

3.3 Evaluation-Based Prioritized Replay (EPR)

As discussed, the principle of prioritized experience replay buffers is to sample states according to their loss, i.e., we more often sample states where the loss is high and less often where the loss is low (see Eq. (7)). Here, our idea is to base the priorities on the outcome of the evaluation instead.

The samples (s, a, r, s') in the replay buffer may be arbitrary and, in particular, may not contain possible initial states. Yet the evaluation is done for initial states only. To be able to judge individual transition samples, we evaluate each sample in terms of the initial state $s_0 \in I$ from which it was generated, i.e., from which the respective training episode started. This arrangement is meaningful as improving the policy for s_0 necessarily involves further training on its successor states. For each transition sample, we store the partition J_i of the initial state s_0 in the replay buffer. The replay priority δ is then set to

$$\delta = (1 - E(J_i) + \epsilon_p)^{\alpha}, \tag{10}$$

where $s_0 \in J_i$ is the initial state of the training episode, and ϵ_p and α have the same functionality as in Eq. (7). After every evaluation stage, we update the priorities of the replay buffer according to Eq. (10). The probability of picking experience (s, a, r, s') during training from the buffer is then proportional to the above replay priority.

3.4 Deep Q-learning with Evaluation Stages

EID is applicable to any (deep) reinforcement learning algorithm, and EPR to any such algorithm using a replay buffer. Here, we implement both methods on top of deep Q-learning [29]. Algorithm 1 shows pseudocode for deep Q-learning

Algorithm 1. Deep Q-learning with Evaluation Stages

```
1: for episodes $i = 0$ to $M - 1$ do
2:     sample $s_0 \in I$ from $\mu$                                   // [DQN, DQNPR, EPR]
3:     sample $s_0 \in I$ according to Equation (9)                     // [EID]
4:     for steps $t = 0$ to $T - 1$ do
5:         with probability $\epsilon$ select random action $a_t \in A(s_t)$
6:         otherwise with probability $1 - \epsilon$ select $a_t = \text{argmax}_{a \in A(s_t)} Q_\theta(s, a)$
7:         execute $a_t$; observe $r_{t+1}$ and $s_{t+1}$
8:         compute $\delta = \begin{cases} \text{constant} & \text{// [DQN, EID]} \\ \text{Equation (7)} & \text{// [DQNPR]} \\ \text{Equation (10)} & \text{// [EPR]} \end{cases}$
9:         store $(s_t, a_t, r_{t+1}, s_{t+1}, \delta)$ in replay buffer $D$
10:        every $C$ steps do
11:            sample a minibatch of samples $(s_j, a_j, r_{j+1}, s_{j+1}, \delta)$ from $D$ w.r.t. $\delta$
12:            set target $y_j = \begin{cases} r_{j+1} & s_{j+1} \text{ is terminal state} \\ r_{j+1} + \gamma \cdot \max_{a'} Q_{\theta'}(s_{j+1}, a') & \text{else} \end{cases}$
13:            perform a gradient descent step on loss $(y_j - Q_\theta(s_j, a_j))^2$
14:            soft-update the network weights $\theta' = (1 - \tau) \cdot \theta + \tau \cdot \theta'$
15:        end every
16:    end for
17:    if $i > P$ then                                                  // [EID, EPR]
18:        every $L$ episodes do                                        // [EID, EPR]
19:            compute and store $E(J_i)$ for all $J_i \in \mathbb{P}$  // [EID, EPR]
20:        end every                                                    // [EID, EPR]
21:    end if                                                           // [EID, EPR]
22: end for
```

with soft updates (denoted DQN) and its previously discussed variant DQNPR, as well as the extensions for EID and EPR.

The unmarked lines in Algorithm 1 are inherited from the original algorithm and are applied in all versions. Lines that are marked differently are only applied in the versions they are marked with, e.g., line 2 is part of DQN, DQNPR and EPR but not of EID. The colored lines mark the extensions of EID (line 3, blue) and the extensions of both EID and EPR (lines 17–21, green). The DSMC-based evaluation stages are inserted after a threshold P of pre-training episodes was met (line 17), and then are repeated every L episodes (line 18). Thus, the total number of training episodes M is given by $M = P + N \cdot L$ where N is the number of performed evaluation stages. The priority δ (marked in orange, line 8) depends on the algorithm:

- Both original deep Q-learning DQN and EID sample uniformly from the replay buffer, so δ is set to a constant value.
- For DQNPR [37], δ is initialized with the maximal temporal difference loss observed throughout the training procedure, and updated in every learning step according to Eq. (7).
- EPR sets the priority to a constant prior to the first ES, and afterwards according to Eq. (10).

4 Case Studies

We next describe the Racetrack benchmark, which we use to evaluate our approach.

4.1 Racetrack

Racetrack originally is a pen and paper game, adopted as a benchmark in the AI community [6,9,27,32,33], particularly for reinforcement learning [5,14,16]. The task is to steer a car on a map towards the goal line without crashing into walls. The map is given by a two-dimensional grid, where each map cell either is free, part of the goal line, or a wall. We assume that initially the car may start on any free map cell with velocity 0 with equal probability (i.e., μ is uniform and I is the set of all non-wall positions with zero velocity).

Figure 2 shows the three maps that we consider in the following. Barto-big (Fig. 2a) was originally introduced by Barto et al. [6]. We designed the other two maps, Maze (Fig. 2b) and River (Fig. 2c), as examples with a more localized structure highlighting the problem of local robustness.

The position and velocity of the car each is a pair of integers, for the x- and y-dimension. In each step, the agent can accelerate the car by at most one unit in each dimension, i.e., the agent can add an element of $\{-1, 0, 1\}$ to each of x and y, resulting in nine different actions. The ground is slippery, meaning that the action might fail, in which case the acceleration/deceleration does not happen and the car's velocity remains unchanged. Each action application fails with a fixed probability that we will refer to as *noise*.

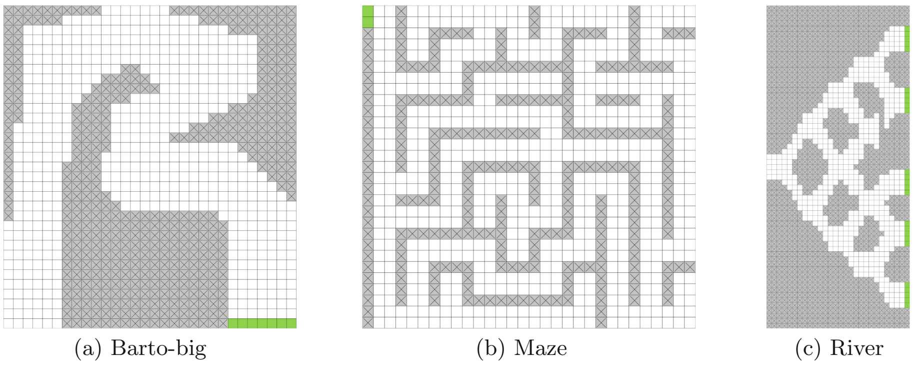

(a) Barto-big (b) Maze (c) River

Fig. 2. Three Racetrack maps, where the goal line is marked in green and wall cells are colored gray. (Color figure online)

The velocity after applying an action defines the car's new position. The car then moves in a straight line from the old position to the new position. If that line intersects with a wall cell, the car crashes and the game is lost. If that line intersects with a goal cell, the game is won. In both cases, the game terminates. We use the following simple reward function:

$$r\left(s \xrightarrow{(ax,ay)} s'\right) = \begin{cases} 100 & \text{if } s' = \top \\ -50 & \text{if } s' = \bot \\ -5 & \text{if } s' = s \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

This reward function is positive if the game was won ($\top$), negative if the game was lost ($\bot$), and slightly negative if the state did not change (incentivizing the agent to not stand still); otherwise no reward signal is given. The incentive to reach the goal as quickly as possible is given through the discount factor γ that is chosen to be smaller than 1, making short-term rewards more important than long-term ones (see Eq. (2)). This reward function encodes the objective to reach the goal as quickly as possible and to not crash into a wall; the concrete values were found experimentally optimizing the performance of the vanilla DQN algorithm.

We remark that one can view the above reward structure as a proxy for the probability to reach the goal. We will consider both perspectives in our experiments, as described next.

4.2 Experiments Setup

The policies (also: *agents*) in our experiments are trained using the different variants of Algorithm 1. Specifically we run DQN and DQNPR, as well as two variants each of our DSMC-based algorithms EID and EPR. The latter variants arise from two different optimization objectives for the evaluation stages in EID and EPR: the expected discounted accumulated reward, which is the same as DRL is trained upon; vs. the probability to reach the goal, as an idealized evaluation objective not suited for training. We denote our algorithms using these objectives with EID^R and EPR^R for the former, and with EID^G and EPR^G for the latter.

For the evaluation stages we use DSMC with an error bound $P(error > \epsilon_{err}) < \kappa$, where $\epsilon_{err} = 0.05$ and $\kappa = 0.05$, i.e., with a confidence of 95% that the error is at most 0.05 [15]. Our partition $\mathbb{P}$ of the initial states I in Racetrack considers each map cell with zero velocity to be a region on its own. For our comparison to be as fair as possible, DQN and DQNPR use the same number of training episodes as the DSMC-based methods, i.e., $M = P + L \cdot N$ (cf. Sect. 3.4).

We use a high noise level, namely 50%, for the Barto-big (Fig. 2a) and river maps (Fig. 2b), to make the decision-making problems challenging. The maze map (Fig. 2c), with its long and narrow paths, is already challenging with much less uncertainty, so we set the noise to 10% there.

All compared approaches use the same neural network structure. We consider multilayer perceptrons (MLPs), aka. feed-forward networks, with a ReLu activation function for every single neuron. We specifically consider the same

NN structure as [15], with input and output layers fixed by Racetrack, and two hidden layers with 64 neurons each.

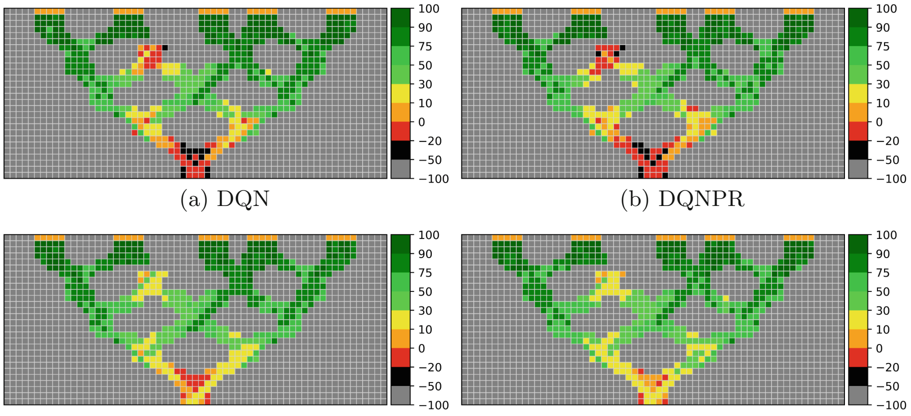
As deep reinforcement learning is known to be sensitive to different random seeds (affecting the exploration of the state space), we perform multiple trainings and report about the average result. Moreover, we fix the random seeds across algorithms in individual runs, so that the first P episodes are equal. The detailed hyperparameter settings can be found in Appendix A.

5 Results

We now analyze whether the inclusion of evaluation stages in the EID and EPR algorithms can improve (i) local robustness and (ii) goal probability performance, compared to the standard algorithms DQN and DQNPR. We first set the evaluation objective to be identical to the expected-reward training objective (EID^R and EPR^R) and analyze whether local robustness is improved; second, we set the evaluation objective to be the goal probability instead (EID^G and EPR^G) and analyze whether the policy's performance for that objective (both on average and local) improves by applying DSMC analysis after training.

5.1 Local Robustness (Deficiency (i))

Consider the heat maps in Fig. 3. For each cell on the map, we plot the expected cumulative discounted reward – the return – when starting from that map cell with zero velocity. In other words, the heat maps have one colored entry for every initial state $s_0 \in I$. We compute the return value for each s_0 using DSMC, with $\epsilon_{err} = 0.01$ and $\kappa = 0.01$, i.e., with a confidence of 99% that the error is at most 0.01.

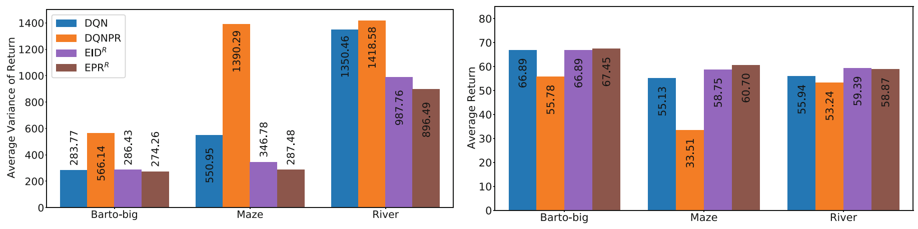


(a) DQN (b) DQNPR

(c) EID^R (d) EPR^R

Fig. 3. Return per map cell on the River map. (Color figure online)

Clearly, the intended improvement of local robustness is achieved by EID^R and EPR^R compared to DQN and DQNPR: the return of the algorithms with evaluation stages is much better in specific areas of the map. This pertains foremost to the bottom end of the map, far away from the goal at the top; and to the "dead-end street" colored red in (a) and (b), where there is no direct connection to the nearest goal and the agents have to temporarily increase the distance to the goal. While the return of EID^R and EPR^R may also seem low in these critical parts, recall that the noise level here is 50% so it is not possible to navigate through this map without a high crash risk.



(a) Return variance (b) Return average

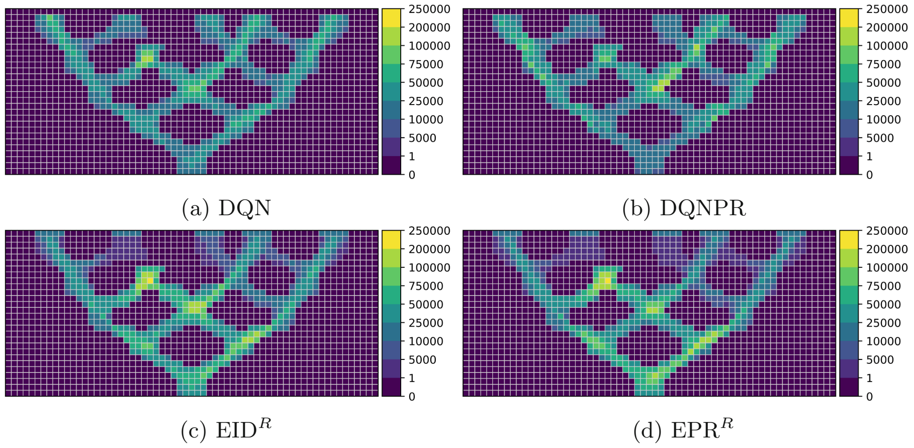
Fig. 4. Variance and average of return on all maps.

Figure 4a summarizes these findings, for all maps, in terms of the variance of the return across the map (the variance of return per map cell).

On the Maze and River maps, the variance of EID^R and EPR^R is much smaller than that of DQN and DQNPR, confirming their improved local robustness. The variance reduction reaches up to about 50% compared to DQN/DQNPR. Among the methods based on DSMC, EPR^R slightly outperforms EID^R. On Barto-big, the variance of DQN is comparable to that of EID^R and EPR^R, which is due to the simpler structure of that map, while for the other two maps the variance is reduced by including DSMC evaluation stages.

Figure 4b shows that, on the Maze and River maps, the improved local robustness also results in somewhat improved average return for EID^R and EPR^R. This shows that evaluation stages can also help with overall performance when challenging local sub-tasks are frequent.

Finally, consider Fig. 5, which confirms that the advantages observed above are indeed due to more intense training in critical parts of the map. We show, for each map cell, the number of times reinforcement learning considered a state where the car was positioned in that cell. The training intensity of DQN is spread fairly homogenously across the map (positions in the dead-end street are seen more often merely because, in any run traversing those, the car needs to turn around). In contrast, EPR^R has a clear focus on the critical parts of the map (which can be seen nicely when comparing Figs. 5c and 5d to Figs. 3a and 3b).



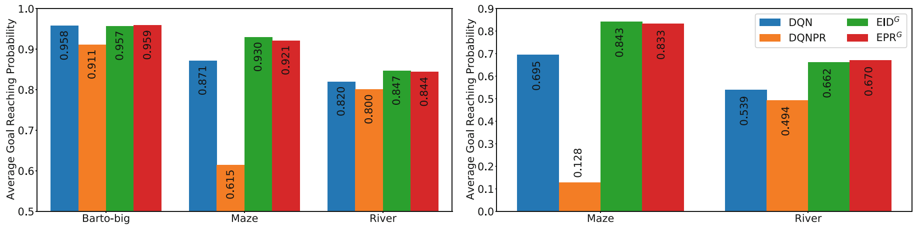
(a) DQN (b) DQNPR

(c) EID^R (d) EPR^R

Fig. 5. Number of times each cell was encountered during training on the River map.

5.2 Fostering Goal Probability (Deficiency (ii))

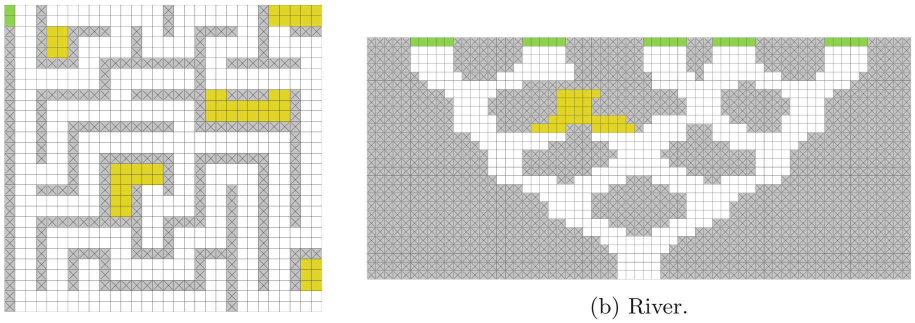
We now turn to deficiency (ii), goal probability performance when training on expected reward. As discussed above, the reward structure is such that goal-reaching is rewarded, but also punishes crashes into the wall. We now show that, indeed, goal-reaching performance can be improved by introducing evaluation stages. EID^G and EPR^G improve the learning signal w.r.t. this objective. In what follows, we compute the goal probability for each map cell – for each initial state $s_0 \in I$ – using DSMC, again with $\epsilon_{err} = 0.01$ and $\kappa = 0.01$, i.e., with a confidence of 99% that the error is at most 0.01.



(a) Across entire map (b) Selected regions (as in Figure 7)

Fig. 6. Average goal probability when training on expected reward, without (DQN and DQNPR) vs. with (EID^G and EPR^G) goal-probability evaluation stages.

Figure 6 shows the corresponding results, (a) for all maps across the entire map, and (b) exemplarily for the Maze map, only for the critical regions. In Fig. 6a, we see that, again, on the Maze and River maps our proposed methods significantly increase the average goal probability. In Barto-big, this does not happen due to the simpler structure of that map.



(a) Maze. (b) River.

Fig. 7. Selected regions (yellow) of the Maze and River maps, as used in Fig. 6b. (Color figure online)

As one would expect, the improvement is higher in critical areas of the maps. To illustrate this, Fig. 6b shows average goal probability for selected regions of the Maze and River maps, as shown in Fig. 7. These regions are the "dead-end streets" which the policy will need to back out from.

6 Conclusion and Future Work

Despite its enormous successes, deep reinforcement learning suffers from important deficiencies in safety-critical systems. Apart from the general inscrutability of neural networks, these include that (i) training on average performance measures lacks local robustness, and that (ii) safety-related objectives like goal probability are sparse and hence not themselves suited for training. We propose to address (i) and (ii) through the incorporation of evaluation stages, which focus the reinforcement learning process on areas of the state space where performance according to an evaluation objective is poor. We observe that such evaluation stages can be readily implemented based on a recently introduced tool for deep statistical model checking [15]. Our experiments on Racetrack, a frequently used benchmark for AI sequential decision-making algorithms [6,9,27,32,33], confirm that this approach can work.

On the algorithmic side, there are various possibilities still to extend our framework, in particular by combining it with other/additional deep reinforcement learning algorithms. Double Q-learning [45], for example, may be promising given the lackluster performance of DQNPR in our experiments. Further, the implementation of our framework on top of policy-based approaches is of interest.

Apart from that, an important direction for future work is the broader empirical exploration of our approach. A straightforward possibility are extensions of Racetrack to include obstacles, traffic, fuel, etc. on a roadmap towards more realistic abstractions of autonomous driving as outlined by [5]. But our approach is of course not limited to Racetrack, and may in principle be applicable in arbitrary contexts where deep reinforcement learning is used. We believe that safety-critical cyber-physical systems should be the prime target, seeing as (i) and (ii) are key in that context, and seeing as the initial state partition required by our approach can be naturally obtained by (coarse discretizations of) physical location. In this context, a particular question to address will be the partition granularity trade-off, between the amount of information available during evaluation stages, and the overhead for conducting them.

A Hyperparameters

Parameter	Description	Value
DQN:		
ϵ_{start}	exploration coefficient in the beginning of the training	0.99
ϵ_{end}	exploration coefficient in the end of the training	0.05
ϵ_{decay}	exponential decay factor of the training coefficient that is applied every episode until ϵ_{end} is reached	0.999
γ	discount factor	0.99
M	number of used training episodes	110000
T	maximal length episodes	100
	learning rate in gradient descend optimization	0.0005
τ	soft update coefficient	0.001
	size of replay buffer	10^8
DQNPR:		
α	prioritization coefficient	1
ϵ_p	minimal priority	10^{-6}
ES-based:		
P	number of pre-training episodes	10000
N	number of evaluation stages	100
L	number of episodes between the evaluation stages	1000
κ	error rate/ half-width parameter during training	0.05
κ	error rate/ half-width parameter during evaluation	0.01
ϵ_{err}	error probability/ confidence during training	0.05
ϵ_{err}	error probability/ confidence during evaluation	0.01

References

1. Agostinelli, F., McAleer, S., Shmakov, A., Baldi, P.: Solving the Rubik's cube with deep reinforcement learning and search. Nat. Mach. Intell. **1**, 356–363 (2019)
2. Alshiekh, M., Bloem, R., Ehlers, R., Könighofer, B., Niekum, S., Topcu, U.: Safe reinforcement learning via shielding. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
3. Amit, R., Meir, R., Ciosek, K.: Discount factor as a regularizer in reinforcement learning. In: International Conference on Machine Learning, pp. 269–278. PMLR (2020)

4. Avni, G., Bloem, R., Chatterjee, K., Henzinger, T.A., Könighofer, B., Pranger, S.: Run-time optimization for learned controllers through quantitative games. In: Dillig, I., Tasiran, S. (eds.) CAV 2019. LNCS, vol. 11561, pp. 630–649. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-25540-4_36
5. Baier, C., et al.: Lab conditions for research on explainable automated decisions. In: Heintz, F., Milano, M., O'Sullivan, B. (eds.) TAILOR 2020. LNCS (LNAI), vol. 12641, pp. 83–90. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73959-1_8
6. Barto, A.G., Bradtke, S.J., Singh, S.P.: Learning to act using real-time dynamic programming. Artif. Intell. **72**(1–2), 81–138 (1995)
7. Bogdoll, J., Hartmanns, A., Hermanns, H.: Simulation and statistical model checking for modestly nondeterministic models. In: Schmitt, J.B. (ed.) MMB&DFT 2012. LNCS, vol. 7201, pp. 249–252. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28540-0_20
8. Bonet, B., Geffner, H.: GPT: a tool for planning with uncertainty and partial information. In: Proceedings of the IJCAI Workshop on Planning with Uncertainty and Incomplete Information, pp. 82–87 (2001)
9. Bonet, B., Geffner, H.: Labeled RTDP: improving the convergence of real-time dynamic programming. In: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 12–21 (2003)
10. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: A statistical model checker for nondeterminism and rare events. In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10806, pp. 340–358. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89963-3_20
11. Ciosek, K., Whiteson, S.: Offer: off-environment reinforcement learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 31 (2017)
12. Frank, J., Mannor, S., Precup, D.: Reinforcement learning in the presence of rare events. In: Proceedings of the 25th International Conference on Machine Learning, pp. 336–343 (2008)
13. Fujita, Y., Nagarajan, P., Kataoka, T., Ishikawa, T.: ChainerRL: a deep reinforcement learning library. J. Mach. Learn. Res. **22**(77), 1–14 (2021)
14. Gros, T.P., Groß, D., Gumhold, S., Hoffmann, J., Klauck, M., Steinmetz, M.: TraceVis: towards visualization for deep statistical model checking. In: Proceedings of the 9th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation. From Verification to Explanation (2020)
15. Gros, T.P., Hermanns, H., Hoffmann, J., Klauck, M., Steinmetz, M.: Deep statistical model checking. In: Proceedings of the 40th International Conference on Formal Techniques for Distributed Objects, Components, and Systems (FORTE 2020) (2020). https://doi.org/10.1007/978-3-030-50086-3_6
16. Gros, T.P., Höller, D., Hoffmann, J., Wolf, V.: Tracking the race between deep reinforcement learning and imitation learning. In: Gribaudo, M., Jansen, D.N., Remke, A. (eds.) QEST 2020. LNCS, vol. 12289, pp. 11–17. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-59854-9_2
17. Gu, S., Holly, E., Lillicrap, T., Levine, S.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: 2017 IEEE International Conference on Robotics and Automation (ICRA), pp. 3389–3396. IEEE (2017)
18. Hare, J.: Dealing with sparse rewards in reinforcement learning. arXiv preprint arXiv:1910.09281 (2019)

19. Hartmanns, A., Hermanns, H.: The modest toolset: an integrated environment for quantitative modelling and verification. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 593–598. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_51
20. Hasanbeig, M., Abate, A., Kroening, D.: Logically-constrained reinforcement learning. arXiv preprint arXiv:1801.08099 (2018)
21. Hinton, G., et al.: Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups. IEEE Signal Process. Mag. **29**(6), 82–97 (2012)
22. Jansen, N., Könighofer, B., Junges, S., Serban, A., Bloem, R.: Safe Reinforcement Learning Using Probabilistic Shields (2020)
23. Junges, S., Jansen, N., Dehnert, C., Topcu, U., Katoen, J.-P.: Safety-constrained reinforcement learning for MDPs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 130–146. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_8
24. Knox, W.B., Stone, P.: Reinforcement learning from human reward: discounting in episodic tasks. In: 2012 IEEE RO-MAN: The 21st IEEE International Symposium on Robot and Human Interactive Communication, pp. 878–885 (2012). https://doi.org/10.1109/ROMAN.2012.6343862
25. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NIPS, pp. 1097–1105 (2012)
26. Liu, C., Xu, X., Hu, D.: Multiobjective reinforcement learning: a comprehensive overview. IEEE Trans. Syst. Man Cybern. Syst. **45**(3), 385–398 (2014)
27. McMahan, H.B., Gordon, G.J.: Fast exact planning in Markov decision processes. In: Proceedings of the International Conference on Automated Planning and Scheduling, pp. 151–160 (2005)
28. Mnih, V., et al.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013). Accessed 15 Sept 2020
29. Mnih, V., et al.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)
30. Nazari, M., Oroojlooy, A., Snyder, L., Takac, M.: Reinforcement learning for solving the vehicle routing problem. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 31, pp. 9839–9849. Curran Associates, Inc. (2018)
31. Ng, A.Y., Harada, D., Russell, S.J.: Policy invariance under reward transformations: theory and application to reward shaping. In: Proceedings of the 16th International Conference on Machine Learning (ICML 1999), pp. 278–287 (1999)
32. Pineda, L.E., Lu, Y., Zilberstein, S., Goldman, C.V.: Fault-tolerant planning under uncertainty. In: Twenty-Third International Joint Conference on Artificial Intelligence, pp. 2350–2356 (2013)
33. Pineda, L.E., Zilberstein, S.: Planning under uncertainty using reduced models: revisiting determinization. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 24 (2014)
34. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming, 1st edn. Wiley, New York (1994)
35. Riedmiller, M., et al.: Learning by playing solving sparse reward tasks from scratch. In: International Conference on Machine Learning, pp. 4344–4353. PMLR (2018)
36. Sallab, A.E., Abdou, M., Perot, E., Yogamani, S.: Deep reinforcement learning framework for autonomous driving. Electron. Imaging **2017**(19), 70–76 (2017)

37. Schaul, T., Quan, J., Antonoglou, I., Silver, D.: Prioritized experience replay. In: Bengio, Y., LeCun, Y. (eds.) 4th International Conference on Learning Representations, ICLR (2016)
38. Schwartz, A.: A reinforcement learning method for maximizing undiscounted rewards. In: Proceedings of the Tenth International Conference on Machine Learning, vol. 298, pp. 298–305 (1993)
39. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: International Conference on Computer Aided Verification, pp. 266–280 (2005)
40. Silver, D., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
41. Silver, D., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)
42. Silver, D., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
43. Stooke, A., Abbeel, P.: rlpyt: a research code base for deep reinforcement learning in Pytorch. arXiv preprint arXiv:1909.01500 (2019)
44. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction, Adaptive Computation and Machine Learning, 2nd edn. The MIT Press, Cambridge (2018)
45. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 30 (2016)
46. Vinyals, O., et al.: Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature **575**, 350–354 (2019)
47. Younes, H.L.S., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking: an empirical study. In: Jensen, K., Podelski, A. (eds.) TACAS 2004. LNCS, vol. 2988, pp. 46–60. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-24730-2_4