Constrained Exploration and Recovery from Experience Shaping

Tu-Hoa Pham, Giovanni De Magistris, Don Joven Agravante,
Subhajit Chaudhury, Asim Munawar and **Ryuki Tachibana**
IBM Research - Tokyo
{pham,giovadem,subhajit,asim,ryuki}@jp.ibm.com, don.joven.r.agravante@ibm.com

Abstract

We consider the problem of reinforcement learning under safety requirements, in which an agent is trained to complete a given task, typically formalized as the maximization of a reward signal over time, while concurrently avoiding undesirable actions or states, associated to lower rewards, or penalties. The construction and balancing of different reward components can be difficult in the presence of multiple objectives, yet is crucial for producing a satisfying policy. For example, in reaching a target while avoiding obstacles, low collision penalties can lead to reckless movements while high penalties can discourage exploration. To circumvent this limitation, we examine the effect of past actions in terms of safety to estimate which are acceptable or should be avoided in the future. We then actively reshape the action space of the agent during reinforcement learning, so that reward-driven exploration is constrained within safety limits. We propose an algorithm enabling the learning of such safety constraints in parallel with reinforcement learning and demonstrate its effectiveness in terms of both task completion and training time.

1 Introduction

Recent work in reinforcement learning has established the potential for deep neural network architectures to tackle difficult control and decision-making problems, such as playing video games from raw pixel information (Mnih et al. 2015), Go (Silver et al. 2016), as well as robot manipulation (Haarnoja et al. 2018) and whole-body control (Peng et al. 2018). Such problems are often characterized by the high dimensionality of possible actions and observations, making them difficult to solve or even intractable for traditional optimization methods. Still, deep reinforcement learning techniques have remained subject to limitations including poor sample efficiency, large requirements in data and interactions with the environment and strong dependency on an appropriately-designed reward signal (Duan et al. 2016). In particular, a considerable challenge towards their applicability to real-world problems is that of safety. Indeed, while deep neural networks can reasonably be employed as *black-box* controllers within simulated or well-controlled environments, limited interpretability and vulnerability to adversarial attacks (Goodfellow, Shlens, and Szegedy 2015; Su, Vargas, and Kouichi 2017) can hinder their deployment to situations where poor decisions can have undesirable consequences for the agent or its environment, e.g., in autonomous driving. For such critical applications, deep neural networks can be used as a specialized service for components offering stability and performance guarantees, e.g., for visual recognition or dynamics prediction in conjunction with model-predictive control (Williams et al. 2017).

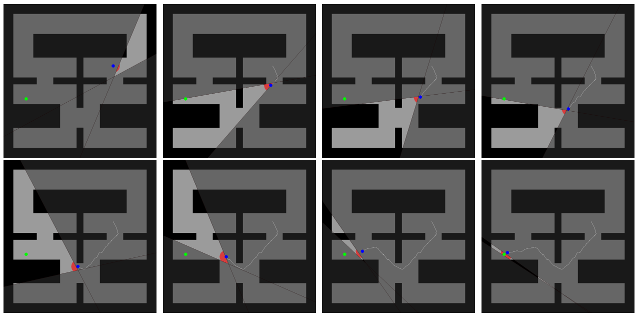


Figure 1: From left to right, top to bottom: consecutive actions taken during reinforcement learning in a maze environment with safety constraints. While in the unconstrained setting, the agent is permitted to take steps anywhere within the red circle, our approach learns safety constraints (shaded area) preventing failure from hitting obstacles (black surfaces).

In the absence of ground-truth physical models, prediction robustness can also be improved when large amounts of data are available or can be generated, e.g., through transfer learning with domain randomization (Tobin et al. 2017). Data can also be used to tackle the limitations of reinforcement learning in terms of training time and reward crafting. When reference trajectories are available, e.g., demonstrated by an expert, it is possible to initialize a control policy by behavioral cloning (Pomerleau 1991), e.g., by training the neural network in a supervised manner using reference states and actions as inputs-outputs. However, behavioral cloning frequently requires tremendous amounts of data that extensively span observation and action spaces. Expert demonstrations were also used in interaction with the reinforcement learning process to accelerate training (Hester et al. 2018). Alternatively, it is possible to use expert data to infer a reward signal that the expert is assumed to be following through inverse reinforcement learning (IRL) (Ng, Russell, and others 2000). However, IRL methods can per-form poorly in the presence of imperfect demonstrations. In addition, even when a reward signal can be estimated, it can remain insufficient to train a control policy by reinforcement learning afterwards. Towards this limitation, (Ho and Ermon 2016) proposed to bypass the reward estimation step by directly training a control policy together with a discriminator classifying state-action pairs as expert-like or not, in a manner analogous to generative adversarial networks (Goodfellow et al. 2014), showing successful imitation from very few expert trajectories on robot control tasks. Other successes have also been obtained in meta-learning frameworks for generalization from single demonstrations (Duan et al. 2017) or reinforcement learning from imperfect demonstrations using multimodal policies (Haarnoja et al. 2017; Gao et al. 2018).

In this work, we propose to use reference demonstrations in a novel manner: not to train a control policy directly, but rather to learn safety constraints towards the completion of a given task. In this direction, we build upon the start of the art in safe reinforcement learning (Section 2). In contrast with traditional imitation learning frameworks, our approach leverages both *positive* and *negative* demonstrations, which we loosely define as aiming to complete and fail the designated task, respectively, yet without need for optimality (e.g., maximum reward or fastest failure).

- We demonstrate that it is possible to automatically learn action-space constraints in a supervised manner even when no ground-truth constraints are available, through the formulation of a loss function acting as a proxy for a convex optimization problem (Section 3).
- Positive and negative reference demonstrations may not be available in many practical problems of interest. Thus, we derive an algorithm, Constrained Exploration and Recovery from Exploration Shaping (CERES), to discover both from parallel instantiations of a reinforcement learning problem while learning safety constraints (Section 4).
- On collision avoidance tasks with dynamics, we show that our approach makes reinforcement learning more efficient, achieving higher rewards in fewer iterations, while also enabling learning from reduced observations (Section 5).

Finally, we discuss the challenges we encountered and future directions for our work (Section 6). To facilitate its reproduction and foster the research in constraint-based reinforcement learning, we make our algorithms public and open-source[1].

2 Background and Motivation

2.1 Reinforcement Learning

We consider an infinite-horizon discounted Markov decision process (MDP) characterized by: S a state domain representing observations available to the agent we seek to control; A an action domain representing how the agent can interact with its environment; $\rho_0 : S \to [0, 1]$ a probability distribution for the initial state; $P : S \times A \times S \to [0, 1]$ a transition probability distribution describing how, from a given state, taking an action can lead to another state; and $R : S \times A \times S \to \mathbb{R}$ a function associating rewards to such transitions. With $\gamma \in [0, 1)$ a discount factor on future reward expectations, we aim to construct a stochastic policy $\pi : S \times A \to [0, 1]$ that maximizes the γ-discounted expected return $\eta(\pi)$:

$$\eta(\pi) = \mathop{\mathbb{E}}_{\tau}\left[\sum_{i=0}^{\infty} \gamma^i R(\mathbf{s}_i, \mathbf{x}_i, \mathbf{s}_{i+1})\right], \qquad (1)$$

with $\tau = (\mathbf{s}_0, \mathbf{x}_0, \mathbf{s}_1, \mathbf{x}_1, \ldots)$ a sequence of states and actions where the initial state $\mathbf{s}_0$ is initialized following ρ_0, and each action $\mathbf{x}_i$ is sampled following $\pi(\cdot|\mathbf{s}_i)$ the control policy given the current state $\mathbf{s}_i$, leading to a new state $\mathbf{s}_{i+1}$ following the transition function $P(\cdot|\mathbf{s}_i, \mathbf{x}_i)$. Through Eq. (1), we seek to maximize not a one-step reward, but rather a reward expectation over time. While S, A, ρ_0, P allow some variation in their implementation (e.g., different resolutions for images as state space), they remain mostly characterized by the considered task. In contrast, the reward function can often be engineered empirically, from intuition, experience, and trial and error. Such a process is ineffective and costly, since evaluating a reward function candidate requires training a policy with it.

We consider deep reinforcement learning in continuous action spaces, in which actions are typically n_{act}-dimensional real-valued vectors, $\mathbf{x} \in A \subset \mathbb{R}^{n_{\text{act}}}$ (e.g., joint commands for a robot arm). Given an n_{obs}-dimensional input state vector $\mathbf{s} \in S \subset \mathbb{R}^{n_{\text{obs}}}$, actions are sampled following a neural network $\mathcal{N}^\pi$ representing the control policy π, $\mathbf{x} \sim \mathcal{N}^\pi(\mathbf{s})$. Multiple methods were developed to tackle such problems, such as Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al. 2015) or Trust Region Policy Optimization (TRPO) (Schulman et al. 2015), which were benchmarked on robot control tasks in (Duan et al. 2016). In this work, we build upon the Proximal Policy Optimization (PPO) algorithm (Schulman et al. 2017) and its OpenAI Baselines reference implementation (Dhariwal et al. 2017), in which the control policy is an n_{act}-dimensional multivariate Gaussian distribution of mean and standard deviation predicted by the neural network $\mathcal{N}^\pi$, trained on-policy by interacting with the environment to collect state-action-reward tuples $(\mathbf{s}_i, \mathbf{x}_i, r_i)_{i=1,T}$ on a T-timestep horizon. Alternative frameworks employing energy-based policies also achieved significant results on improved exploration and skill transfer between tasks (Haarnoja et al. 2017; 2018).

2.2 Safe Reinforcement Learning

While failure is most often permissible in simulated environments, real-world applications often come with requirements in terms of safety, for both the artificial agent and its environment. Indeed, poor decisions may have undesirable consequences, both in the physical world (e.g., an autonomous vehicle colliding with another vehicle or person) and within information systems (e.g., algo trading). Thus, the topic of safe reinforcement learning has been the subject of considerable research from multiple perspectives (Garcıa and Fernández 2015). From the deep reinforcement learning domain, (Achiam et al. 2017) recently proposed a trust region method,

[1] https://www.github.com/IBM/constrained-rl

named Constrained Policy Optimization (CPO), enabling the training of control policies with near-satisfaction of given, known safety constraints. Towards real-world applications, (Pham, De Magistris, and Tachibana 2018) proposed to combine the TRPO reinforcement learning algorithm with an optimization layer that takes as input an action predicted by a neural network policy and correct it to lie within safety constraints via convex optimization. There again, safety constraints are required to be specified in advance. Namely, given a state $\mathbf{s}$, an action is sampled from a neural network policy as $\widetilde{\mathbf{x}} \sim \mathcal{N}^{\pi}(\mathbf{s})$. Instead of directly executing $\widetilde{\mathbf{x}}$ onto the environment, as the neural network has no explicit safety guarantee, it is first corrected by solving the following quadratic program (QP) (Mattingley and Boyd 2012):

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^{n_{\text{act}}}}{\operatorname{argmin}} \left\{ \|\mathbf{x} - \widetilde{\mathbf{x}}\|^2 \ \text{ such that } \ \mathbf{G}\mathbf{x} \leq \mathbf{h} \right\}, \qquad (2)$$

with $\mathbf{G}$ and $\mathbf{h}$ linear constraint matrices of respective size $n_{\text{in}} \times n_{\text{act}}$ and $n_{\text{in}} \times 1$ describing the range of possible actions. The closest action satisfying these constraints, $\mathbf{x}^*$, is then executed in the environment. While in Eq. (2), $\mathbf{G}$ and $\mathbf{h}$ are assumed provided by the user, in our work, we instead propose to learn them. This is a rather unexplored idea, since safety constraints can sometimes be constructed in a principled way, e.g., using the equations of physics in robotics. However, doing so is often cumbersome and possibly imprecise, as it depends on the availability of an accurate model of the agent and its environment. In contrast, our approach operates in a complete model-free fashion and is able to learn from direct demonstrations (possibly generated from scratch), without need for prior knowledge.

Other evidence of reinforcement learning acceleration through improved exploration was presented in (Wachi et al. 2018), where safety constraints where modelled with Gaussian processes. From the perspective of planning, (Cserna et al. 2018) defined safety not as a numerical quantity as in the previous works, but through the notion of avoiding dead ends, from which the task can no longer be completed. In our work, we similarly define *negative* demonstrations $d \in \mathcal{D}^-$ as state-action couples $d = (\mathbf{s}, \mathbf{x})$ such that taking $\mathbf{x}$ from $\mathbf{s}$ inevitably leads to failure: either directly (e.g., the agent immediately crashes against a wall) or because recovery is no longer possible after taking $\mathbf{x}$ (e.g., still accelerating despite passing a minimum braking distance). Conversely, we define *positive* demonstrations $d \in \mathcal{D}^+$ as state-action couples such that the agent can still recover from the resulting state (e.g., starting to decelerate before passing the minimum braking distance). The set of demonstrations $\mathcal{D}^*$ that are neither known to be positive or negative are called *uncertain*. Although determining beyond doubt whether an uncertain demonstration is positive or negative may be intractable in many cases, we propose a heuristic approach to sample and classify such demonstrations through a specialized reinforcement learning process. Our approach is thus related to that of (Eysenbach et al. 2018), where a reset policy was learned to return the environment to a safe state for future attempts. In (Pinto et al. 2017), increased robustness was achieved by training the control policy together with an adversary learning to produce optimal perturbations. While such perturbations can be used as negative demonstrations, we seek to collect a variety of such examples without need for optimality (e.g., any action leading an agent to collide against a wall, without necessary inducing the greatest impact). Finally, although not reinforcement learning, we were also inspired by the work of (Gandhi, Pinto, and Gupta 2017), where negative demonstrations were collected by purposely crashing a drone into surrounding objects to learn whether a direction is safe to fly to as a simple binary classification problem.

3 Learning Action-Space Constraints from Positive and Negative Demonstrations

3.1 Definitions

State-dependent action-space constraints Let $(c_i)_{i\in[1,n_{\text{in}}]}$ denote a set of n_{in} constraints functions operating on actions $\mathbf{x} \in \mathbb{R}^{n_{\text{act}}}$, of the general form:

$$c_i(\mathbf{x}) \leq 0, \quad i \in [1, n_{\text{in}}]\,. \qquad (3)$$

In general, the constraint functions c_i can take different forms but are typically real-valued, e.g., $c_i(\mathbf{x}) = \|\mathbf{x}\|_2 - 1$, the second-order cone inequality constraining $\mathbf{x}$ to be of $\mathcal{L}^2$ norm 1 or less. We consider in particular the case of linear inequalities, e.g., $2x_0 - x_1 \leq 3$, parameterized by a row vector $\mathbf{g}_i$ of size n_{act} and a scalar h_i such that:

$$c_i(\mathbf{x}) = \mathbf{g}_i\mathbf{x} - h_i, \quad i \in [1, n_{\text{in}}]\,. \qquad (4)$$

With $\mathbf{G} = [\mathbf{g}_1, \dots, \mathbf{g}_{n_{\text{in}}}]^T$ and $\mathbf{h} = [h_1, \dots, h_{n_{\text{in}}}]^T$ the constraint matrices of respective size $n_{\text{in}} \times n_{\text{act}}$ and $n_{\text{in}} \times 1$, Eq. (3) takes the familiar form $\mathbf{G}\mathbf{x} \leq \mathbf{h}$ of Eq. (2), with inequalities considered row-wise. We are interested in estimating such constraint matrices as functions of state vectors $\mathbf{s} \in \mathbb{R}^{n_{\text{obs}}}$, e.g., as outputs of a neural network $\mathcal{N}^C$:

$$\mathbf{G}^{\mathbf{s}}\mathbf{x} \leq \mathbf{h}^{\mathbf{s}}, \ \text{ with } \ (\mathbf{G}^{\mathbf{s}}, \mathbf{h}^{\mathbf{s}}) = \mathcal{N}^C(\mathbf{s}). \qquad (5)$$

Formally, we thus consider constraints that operate on the action domain and depend on the current state (e.g., an autonomous vehicle may not accelerate more than a given rate – action constraint – if another vehicle is less than a given distance ahead – current state). Eq. (4) can thus be rewritten:

$$c_i^{\mathbf{s}}(\mathbf{x}) = \mathbf{g}_i^{\mathbf{s}}\mathbf{x} - h_i^{\mathbf{s}}, \quad i \in [1, n_{\text{in}}]\,, \qquad (6)$$

with $c_i^{\mathbf{s}}(\mathbf{x}) \leq 0$ when the constraint is satisfied, and $c_i^{\mathbf{s}}(\mathbf{x}) > 0$ when it is violated. Given a demonstration $d = (\mathbf{s}, \mathbf{x})$, wethen define, for each constraint $i \in [1, n_{\text{in}}]$, a satisfaction margin $M_i^S(\mathbf{s}, \mathbf{x})$ and a violation margin $M_i^V(\mathbf{s}, \mathbf{x})$:

$$M_i^S(\mathbf{s}, \mathbf{x}) = \max\left(0, -c_i^{\mathbf{s}}(\mathbf{x})\right) = \operatorname{ReLU}\left(-c_i^{\mathbf{s}}(\mathbf{x})\right), \qquad (7)$$

$$M_i^V(\mathbf{s}, \mathbf{x}) = \max\left(0, c_i^{\mathbf{s}}(\mathbf{x})\right) = \operatorname{ReLU}\left(c_i^{\mathbf{s}}(\mathbf{x})\right), \qquad (8)$$

with $\max$ the maximum operator, which for comparisons with zero can be represented by ReLU the rectified linear unit. Thus, $M_i^S(\mathbf{s}, \mathbf{x})$ (resp. $M_i^V(\mathbf{s}, \mathbf{x})$), is positive if the i-th constraint is satisfied (resp. violated), and zero otherwise. Finally, given a set of known positive and bad demonstrations, we associate to each an indicator $\delta^+_{\mathbf{s},\mathbf{x}}$ equal to 1 if $(\mathbf{s}, \mathbf{x})$ is a positive demonstration and 0 otherwise.

3.2 Constraint Training Loss

In this Section, we assume the availability of state-action demonstrations $(\mathbf{s}, \mathbf{x})$ along with associated indicators $\delta^{+}_{\mathbf{s},\mathbf{x}}$ (e.g., provided by a human expert). We then seek to construct constraint functions $(c_i)_{i \in [1, n_{\text{in}}]}$ that satisfy the following. If $(\mathbf{s}, \mathbf{x})$ is a positive demonstration, then we want *all* constraints to be satisfied:

$$\delta^{+}_{\mathbf{s},\mathbf{x}} = 1 \implies \forall i \in [1, n_{\text{in}}], c^{\mathbf{s}}_i(\mathbf{x}) \leq 0. \tag{9}$$

Having all constraints satisfied is equivalent to having none violated. Using the violation margin defined in Eq. (8) yields:

$$\delta^{+}_{\mathbf{s},\mathbf{x}} = 1 \implies \forall i \in [1, n_{\text{in}}], M^V_i(\mathbf{s}, \mathbf{x}) = 0. \tag{10}$$

Since by definition, all margins are non-negative, we get:

$$\delta^{+}_{\mathbf{s},\mathbf{x}} = 1 \implies \max_{i \in [1, n_{\text{in}}]} \left\{ M^V_i(\mathbf{s}, \mathbf{x}) \right\} = 0. \tag{11}$$

Conversely, if $(\mathbf{s}, \mathbf{x})$ is a negative demonstration, we want *at least one* constraint to be violated:

$$\delta^{+}_{\mathbf{s},\mathbf{x}} = 0 \implies \exists i \in [1, n_{\text{in}}], c^{\mathbf{s}}_i(\mathbf{x}) > 0. \tag{12}$$

This amounts to having at least one constraint of zero satisfaction margin, while others can be strictly positive:

$$\delta^{+}_{\mathbf{s},\mathbf{x}} = 0 \implies \min_{i \in [1, n_{\text{in}}]} \left\{ M^S_i(\mathbf{s}, \mathbf{x}) \right\} = 0. \tag{13}$$

Thus, we can define a constraint loss $\mathcal{L}^C$ comprising the maximum violation for positive demonstrations following Eq. (11) and the mimimum satisfaction for negative demonstrations following Eq (13):

$$\begin{aligned} \mathcal{L}^C\left(\mathbf{s}, \mathbf{x}, \delta^{+}_{\mathbf{s},\mathbf{x}}\right) = \delta^{+}_{\mathbf{s},\mathbf{x}} & \max_{i \in [1, n_{\text{in}}]} \left\{ M^V_i(\mathbf{s}, \mathbf{x}) \right\} \\ & + \left(1 - \delta^{+}_{\mathbf{s}}(\mathbf{x})\right) \min_{i \in [1, n_{\text{in}}]} \left\{ M^S_i(\mathbf{s}, \mathbf{x}) \right\} \end{aligned} \tag{14}$$

Backtracing from Eq. (14) to Eq. (5) shows that $\mathcal{L}^C\left(\mathbf{s}, \mathbf{x}, \delta^{+}_{\mathbf{s},\mathbf{x}}\right)$ is computed as a succession of differentiable operations from $\left(\mathbf{s}, \mathbf{x}, \delta^{+}_{\mathbf{s},\mathbf{x}}\right)$. As the constraint matrices are computed in particular from $\mathbf{s}$ being fed through the constraint network $\mathcal{N}^C$, it can thus be trained in a supervised manner by minimizing $\mathcal{L}^C$ as a training loss, using existing stochastic optimization methods such as Adam (Kingma and Ba 2014). Still, some considerations remain.

3.3 Optimizing the Constraint Loss in Practice

Constraint Normalization In practice, directly minimizing the loss function of Eq. (14) does not suffice to yield useful constraints in practice. Indeed, from the definition of satisfaction and violation margins, it appears that $\mathbf{G}^{\mathbf{s}} = \mathbf{0}$ and $\mathbf{h}^{\mathbf{s}} = \mathbf{0}$ yields a trivial minimum for $\mathcal{L}^C$. In fact, simply having $\mathbf{G}^{\mathbf{s}} = \mathbf{0}$ results in the optimization problem being ill-defined. Considering individual constraint parameters $(\mathbf{g}^{\mathbf{s}}_i, h^{\mathbf{s}}_i)$, we can instead observe that when $\mathbf{g}^{\mathbf{s}}_i$ is non-zero, $\mathbf{g}^{\mathbf{s}}_i\mathbf{x} - h^{\mathbf{s}}_i = 0$ is the equation of a hyperplane in $\mathbb{R}^{n_{\text{act}}}$ (i.e., a line in 2D action space, a plane in 3D action spaces, etc.), of normal $\mathbf{g}^{\mathbf{s}}_i$ itself. Geometric considerations then yield that $\frac{\mathbf{g}^{\mathbf{s}}_i\mathbf{x} - h^{\mathbf{s}}_i}{\|\mathbf{g}^{\mathbf{s}}_i\|}$ is the signed distance between $\mathbf{x}$ and the constraint hyperplane. It thus appears that having each row $\mathbf{g}^{\mathbf{s}}_i$ of the predicted constraint matrix $\mathbf{G}^{\mathbf{s}}$ to be of unit norm would be practical, for both avoiding trivial optima while maintaining geometric interpretability. One possibility is to systematically renormalize satisfaction and violation margins by division with the norm of each $\mathbf{g}^{\mathbf{s}}_i$ post-prediction, within Eqs. (7) and (8). However, we noted that doing so could result in two problems in particular: the neural network predictions growing indefinitely large as they are normalized within the training loss, or conversely decreasing in norm such that $\|\mathbf{g}^{\mathbf{s}}_i\|$ eventually becomes close to zero, causing numerical errors.

Unit constraint matrices Instead of re-normalizing row constraint matrices *a posteriori*, we adopt an alternative formulation ensuring that they are of unit norm in the first place. Recalling that each row can be interpreted as a unit vector in $\mathbb{R}^{n_{\text{act}}}$, we have the constraint neural network predict it in generalized spherical coordinates, representing n_{act}-dimensional vectors in Cartesian coordinates as a radius r and $n_{\text{act}} - 1$ angular coordinates $\phi_1, \ldots, \phi_{n_{\text{act}}-1}$. For example, 2D vectors in Cartesian coordinates can be computed from polar coordinates (r, ϕ) as $x_0 = r\cos(\phi), x_1 = r\sin(\phi)$, with analogous formulas for generalized n_{act}-dimensional spheres. By simply setting the radius to 1, any combination of angles in $\mathbb{R}^{n_{\text{act}}-1}$ produces in a unit vector in $\mathbb{R}^{n_{\text{act}}}$. We then change the output layer of the neural network $\mathcal{N}^C$ so that it predicts n_{act} parameters for each constraint i: $n_{\text{act}} - 1$ spherical coordinates for $\mathbf{g}_i$ and the scalar h_i. The transformation from spherical to Cartesian coordinates only involving cosine and sine functions, the differentiability of the loss function is preserved.

Avoiding constraint incompatibility While constraint satisfaction and violation terms appear together in Eq. (14), they may not be optimized on partially overlapping demonstrations, e.g., a positive and negative demonstration sharing the same state (one action leading to failure and the other not). As isolated demonstrations do not suffice to cleanly separate action-spaces for any state, it is possible that the neural network produces constraints that minimize the training loss $\mathcal{L}^C$ but are incompatible with each other. For example, it is not possible to simultaneously satisfy $x_0 <= 1$ and $x_0 >= 2$. Instead, we would like to ensure that the domain described by $\mathbf{G}^{\mathbf{s}}\mathbf{x} \leq \mathbf{h}^{\mathbf{s}}$ never boils down to the empty set. Remark that if $\mathbf{h}^{\mathbf{s}} \geq \mathbf{0}$, then the optimization problem is always solvable, since the valid domain now contains at least $\mathbf{x} = \mathbf{0}$. While it is straightforward to enforce $\mathbf{h}^{\mathbf{s}} \geq \mathbf{0}$, e.g., by passing it through a ReLU operation, having $\mathbf{x} = \mathbf{0}$ as default fallback action may not always be safe in practice. Instead of $\mathbf{0}$, given an arbitrary point $\hat{\mathbf{x}} \in \mathbb{R}^{n_{\text{act}}}$, it is possible to parameterize constraints such that $\hat{\mathbf{x}}$ always satisfies $\mathbf{G}^{\mathbf{s}}\mathbf{x} \leq \mathbf{h}^{\mathbf{s}}$, by decomposing the right hand-side into $\mathbf{h} = \mathbf{G}^{\mathbf{s}}\hat{\mathbf{x}} + \mathbf{h}^{\mathbf{s}}_{+}$, with $\mathbf{h}^{\mathbf{s}}_{+} \geq \mathbf{0}$. While $\hat{\mathbf{x}}$ can be fixed manually, it can also be considered as an *interior point* that can be learned and shared with each individual constraint. Finally, the bounds of $\mathbf{h}^{\mathbf{s}}_{+}$ can also be set to guarantee, e.g., a minimum or maximum distance between constraints and the interior point $\hat{\mathbf{x}}$. In the following, we set the minimum value of $\mathbf{h}^{\mathbf{s}}_{+}$ to $10\,\%$ of half the action space

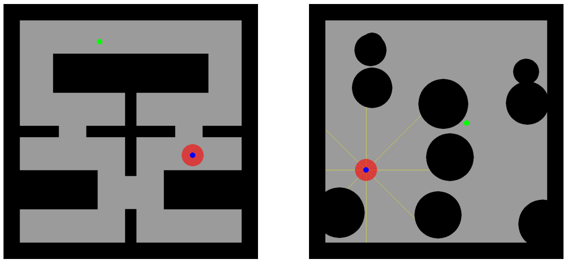


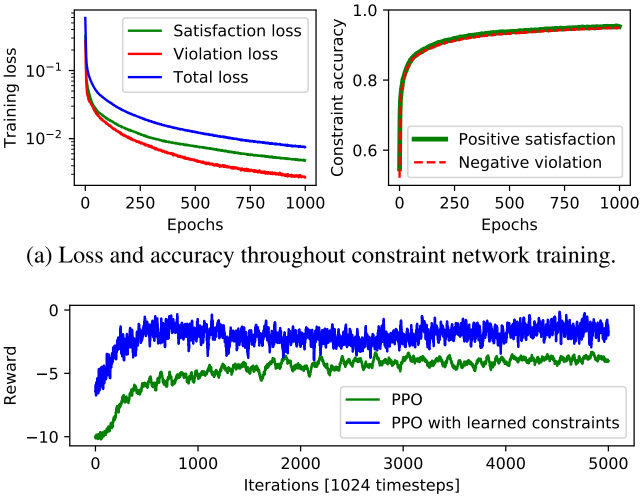
Figure 2: An agent (blue) is tasked to reach a target (green), while avoiding walls and obstacles (black) in a static maze (left) or with random obstacles (right) with shortest distances in eight directions (yellow). The agent can be controlled with position (within the red area) or force commands.

range and its maximum value to half the action space range directly, so that constraint satisfaction never becomes trivial while guaranteeing a minimum exploration volume.

3.4 Application

We consider a task consisting in controlling an agent to reach a target point in a maze-like environment, see Fig. 2 (left). Both the agent and the target are represented by circles of diameter 0.05. Throughout its motion, the agent has to avoid certain areas of the map: the external bounds of the world represented by a square of side 2, $[-1, 1]^2$, and holes in the ground, represented by black surfaces. At the beginning of each episode, agent and target positions are randomly sampled within the allowed surface. At each timestep, the state vector comprises the position of the target and that of the agent. The agent can then make a 2D motion as action vector $\mathbf{x} = (\Delta x, \Delta y) \in [-0.1, 0.1]^2$. If the norm of $\mathbf{x}$ is within a maximum step size $\Delta_m = 0.1$, the position is directly incremented by it, else it is clipped to lie within the circular movement range of radius Δ_m. Each action thus results in an updated state and a reward signal of the form $r = r_{\text{fail}} + r_{\text{goal}} + r_{\text{dist}} + r_{\text{alive}}$, with $r_{\text{fail}} = -10$ a penalty when reaching the border of the world or the central hole, $r_{\text{goal}} = +10$ a bonus when reaching the target, $r_{\text{dist}} = -0.01 * \|\mathbf{P}_T - \mathbf{P}\|$ a reward on the distance between the agent of the target (increasing towards zero as the distance decreases), and $r_{\text{alive}} = -0.01$ a constant penalty per timestep encouraging rapid completion of the task. The episode ends when $T = 100$ timesteps have passed or when either r_{fail} or r_{goal} occurs.

We then collect a set of expert demonstrations by having a human user directly controlling the agent with the mouse, without specific instructions on how to reach the goal (e.g., shortest path possible). We collect 500 such trajectories and take them as our set of positive demonstrations $\mathcal{D}^+$. As this environment does not involve complicated dynamics for the agent, we can define bad actions as those immediately leading to fail the task. We thus iterate through each positive demonstration and sample actions along the circular action range of radius Δ_m. State-action couples leading to task failure are then taken as negative demonstrations. As an additional heuristic, we also consider the expert path, reversed, as negative demonstrations, i.e., if a positive demonstration $(\mathbf{x}_i, \mathbf{s}_i)$ leads to the state $\mathbf{s}_{i+1}$, then we take $(\mathbf{s}_{i+1}, -\mathbf{x}_i)$ as negative demonstration. Note that these heuristics are only applicable due to the simplicity of the environment and its dynamics. We discuss their automatic discovery in the next Section. Overall, we thus collect a set of 23222 demonstrations: 7228 positive and 15994 negative.



(a) Loss and accuracy throughout constraint network training.

(b) Reinforcement learning with and without learned constraints.

Figure 3: Minimizing the constraint loss indeed results in correct separation for good and negative demonstrations (a). After training, the learned constraints can guide exploration during reinforcement learning to achieve higher rewards (b).

We then train a constraint network $\mathcal{N}^C$ to predict $n_{\text{in}} = 2$ constraints on the action space of the agent when exploring the maze. We minimize the constraint loss $\mathcal{L}^C$ using the Adam optimizer, on mini-batches of size 64 comprising 32 positive demonstrations and 32 negative demonstrations each. In doing so, each training epoch consists in iterating through the 15994 negative demonstrations exactly once, while each of the 7228 positive demonstration appears on average 2.2 times per epoch. Alternatively, we could weigh violation and satisfaction losses differently in Eq. (14), e.g., in function of their proportion in the total dataset. We depict the resulting training loss in Fig. 3a. By counting how many positive (resp. negative) demonstrations actually satisfy (resp. violate) the predicted constraints after each training epoch, we empirically verify that the proposed loss $\mathcal{L}^C$ constitutes a representative proxy to learn constraints from demonstrations only. Once $\mathcal{N}^C$ is done training, we embed it within a reinforcement learning process to predict constraints from states encountered during exploration and thus guide the behavior of the agent. Fig. 3b illustrates that this enables both starting from higher rewards, since penalty-heavy collisions are avoided, and reaching a higher reward after training. We depict a full trajectory along with a visualization of action-space constraints in Fig. 1.

4 Constrained Exploration and Recovery from Experience Shaping

4.1 Overview

We established in Section 3 that it is possible to learn action-space constraints as functions of states to guide exploration during reinforcement learning, given a set of positive and negative demonstrations. However, the acquisition of such demonstrations is often problematic on problems of practical interest. First, one cannot always assume the availability of a human expert, e.g., for tasks that humans struggle to complete and look to automate, such as robotic tasks involving high payloads or requiring sub-millimeter accuracy. Second, even when positive demonstrations are available, there may not be clear heuristics to infer negative from positive demonstrations (e.g., by "reversing" them). Third, direct sampling and success-failure evaluation can quickly become intractable on high-dimensional state and action spaces. Finally, even on low-dimensional domains, one may not be able to evaluate an action in a single step. Instead, the effects of a given action may only appear many steps later, mitigated by other events that happened in between. As a result, it is essential to derive an algorithm enabling the discovery and identification of positive and negative demonstrations starting from scratch.

We propose to do so through the reinforcement learning setting. First, we train a *direct* control policy $\mathcal{N}_d^\pi$ that learns to complete the task. After each trajectory, state-action couples are evaluated to determine if they can be labeled positive or negative. In this framework, we consider a demonstration as positive if from the successor step, there exists a trajectory that does not lead to failure within n_s steps, with n_s a hyper-parameter to be chosen in function of dynamics of the task. Conversely, we consider a demonstration as negative if the resulting state only leads to failure within n_s. At this stage, only the final demonstration can confidently be labeled negative, if the trajectory terminates with failure, while only the first demonstrations, of remaining trajectory length greater than n_s, can confidently be labeled positive.

4.2 Demonstration Sorting by Learning Recovery

The second part of our algorithm thus consists in transferring the uncertain demonstrations sampled by the direct policy to a *recovery* control policy $\mathcal{N}_r^\pi$ that learns to recover from such uncertain states. Namely, training of $\mathcal{N}_r^\pi$ involves resetting episodes only to uncertain states visited by the direct policy. In addition, the reward signal r_r used to train $\mathcal{N}_r^\pi$ is simplified to being equal to $+1$ if the agent is still alive at each timestep, and $-n_s$ if it fails the task. If the recovery agent is still active after n_s, the demonstration leading to the episode's starting state (sampled from the direct policy) is labeled as positive, recursively with all predecessor demonstrations. Conversely, if recovery was unsuccessful for a chosen number of attempts n_a, the starting direct demonstration is labeled as negative, along with all successor demonstrations. We remark that, when evaluating trajectories, it is useful to start from the middle as the characterization of a given demonstration affects that of either all its predecessors or all its successors, thus halving the search space each time. Overall, the positive and negative demonstrations collected from both direct and recovery policies can then be used to train a constraint network $\mathcal{N}^C$ to guide the exploration for $\mathcal{N}_d^\pi$, and optionally $\mathcal{N}_r^\pi$.

In summary, given an environment $\mathcal{E}^d$ on which we seek to train a direct policy $\mathcal{N}_d^\pi$, our approach necessitates the following adjustments in creating a recovery environment $\mathcal{E}^r$ to train π^r: 1. a simplified reward that only penalizes task failure, 2. the availability of success and failure flags regarding the final action prior to episode termination, and 3. a function restoring the environment to chosen states. While 3 may appear rather restrictive, the idea of restoring reference states was also used to guide reinforcement learning for whole-body robot control in (Peng et al. 2018). Alternatively, when such a restoration function is unavailable but the environment can be finely controlled, we could consider simply resetting it to reference states by replaying a set number of demonstrations from the sampled direct trajectories. Finally, 2 is necessary to confidently classify the final demonstration, as early episode termination can occur from reasons besides failure (negative), such as completing the task (positive) or just reaching a maximum number of timesteps (uncertain).

4.3 Detailed Algorithm

Conventionally, in the on-policy reinforcement learning setting, states, actions, rewards and other relevant quantities (e.g., value, termination, etc.) are collected as trajectories τ_{PO} following predictions from the neural network policy that are then executed onto the environment in order to update a policy network $\mathcal{N}^\pi$, through the use of a UPDATEPOLICY method e.g., PPO. In CERES, described in Fig. 4, positive, negative, and uncertain demonstrations are sampled together with τ_{PO} within a SAMPLE method. Each time a state-action demonstration $(\mathbf{s}, \mathbf{x})$ is labeled as positive or negative, we store it together with the associated indicator $\delta^+_{\mathbf{s},\mathbf{x}}$, into an experience replay buffer $\mathcal{B}$, then used to iteratively train a constraint network $\mathcal{N}^C$ with an UPDATECONSTRAINTS method following Section 3. In parallel with each policy update, Uncertain trajectories are also transfered from direct to recovery environments to serve as episode initialization states.

The SAMPLE method is further described in Fig. 5. We highlight in particular the following. On line 9, raw actions predicted by the policy network are corrected using a method CONSTRAIN implementing the quadratic program of Eq. (2). Then, on line 11, it is the initial prediction that is used for policy update and not the corrected action, as training is done on-policy. Still, on line 12 it is the corrected action that is used as reference demonstration, since it is the action that is effectively performed onto the environment. Finally, given such unlabeled demonstrations, we sort them as positive, negative and uncertain through a procedure EVALUATEDEMOS implementing the logic described in Section 4.2.

5 Experiments

5.1 Practical Implementation

We implement the constraint learning framework and the CERES algorithm within Tensorflow, while building upon the OpenAI Baselines with PPO as reinforcement learning method for training direct and recovery agents. Preliminary

```
procedure CERES($\mathcal{E}^d, \mathcal{E}^r, \mathcal{N}_d^\pi, \mathcal{N}_r^\pi, \mathcal{N}^C$)
    $\mathcal{N}_d^\pi$.initialize(), $\mathcal{N}_r^\pi$.initialize(), $\mathcal{N}^C$.initialize()
    $\mathcal{B} = \{\}$        ▷ Experience replay buffer start empty
    for $i_{\text{iter}} = 1$ to $n_{\text{iter}}$ do
        $\tau_{\text{PO}}^d, D_+^d, D_-^d, D_*^d = \textsc{Sample}(\mathcal{E}^d, \mathcal{N}_d^\pi, \mathcal{N}^C)$
        $\textsc{UpdatePolicy}(\mathcal{N}_d^\pi, \tau_{\text{PO}}^d)$
        $\tau_{\text{RL}}^r, D_+^r, D_-^r, D_*^r = \textsc{Sample}(\mathcal{E}^r, \mathcal{N}_r^\pi, \mathcal{N}^C)$
        $\textsc{UpdatePolicy}(\mathcal{N}_r^\pi, \tau_{\text{PO}}^r)$
        $\mathcal{B}$.append($D_+^d, D_-^d, D_+^r, D_-^r$)
        $\textsc{UpdateConstraints}(\mathcal{N}^C, \mathcal{B})$
        $\mathcal{E}^r$.add_for_recovery$(D_*^d)$
    end for
    return trained $\mathcal{N}_d^\pi, \mathcal{N}_r^\pi, \mathcal{N}^C$
end procedure
```

Figure 4: In CERES, a direct control policy is trained together with a recovery policy, yielding positive and negative demonstrations to learn and apply action-space constraints.

```
 1: procedure SAMPLE($\mathcal{E}, \mathcal{N}^\pi, \mathcal{N}^C$)
 2:     $\tau_{\text{PO}} = ()$                 ▷ Trajectories for policy update
 3:     $\tau_C = ()$ ▷ Trajectories for demonstration evaluation
 4:     $\mathbf{s} = \mathcal{E}$.reset(); end = false        ▷ Get initial state
 5:     while not end do
 6:         $S$.add($\mathbf{s}$)                ▷ Store current state
 7:         $\mathbf{x} = \mathcal{N}^\pi(\mathbf{s})$                 ▷ Predict action
 8:         $\mathbf{G}^{\mathbf{s}}, \mathbf{h}^{\mathbf{s}} = \mathcal{N}^C(\mathbf{s})$         ▷ Predict constraints
 9:         $\mathbf{x}^* = \textsc{Constrain}(\mathbf{x}, \mathbf{G}^{\mathbf{s}}, \mathbf{h}^{\mathbf{s}})$ ▷ Correct action
10:         $\widetilde{\mathbf{s}}, r$, end, info $= \mathcal{E}$.do$(\mathbf{x}^*)$      ▷ Play corrected
11:         $\tau_{\text{PO}}$.append($\mathbf{s}, \mathbf{x}, r$, end)
12:         $\tau_C$.append($\mathbf{s}, \mathbf{x}^*$, info)
13:     end while
14:     $D_+, D_-, D_* = \textsc{EvaluateDemos}(\tau_C)$
15:     return $\tau_{\text{PO}}, D_+, D_-, D_*$
16: end procedure
```

Figure 5: Trajectories are sampled for policy and constraint learning from positive, negative, uncertain demonstrations.

experiments showed that since constraint predictions can be rather inaccurate over the first iterations, as labeled demonstrations are still few, it is possible to only correct the action prediction with a certain probability in Fig. 5, line 9, and otherwise play the predicted action directly in the environment. We empirically found that an appropriate metric for the constraint activation probability is the percentage of actions that are correctly separated by the predicted constraints (i.e., the proportion of positive actions satisfying the predicted constraints and negative actions violating them). We also obtained good results by only constraining the direct policy, enabling a more diverse range of sampled actions to learn recovery, prior to training the constraint network.

5.2 Obstacle Avoidance with Dynamics

While the example considered in Section 3.4 was limited by fixed safe domains and position control for the agent, we now consider the case where hole placement is randomized at each episode, see Fig. 2 (right), and where the agent can be controlled with force commands. In the latter case, it is now insufficient to evaluate demonstrations as good or bad from a single step, as the agent can no longer stop instantly if it is travelling at maximum speed. In addition to the positions of the agent and the target, the state vector now includes the current linear velocity of the agent and its distance to surrounding obstacles akin to LIDAR systems, along 8 regularly-spaced beams starting from its center.

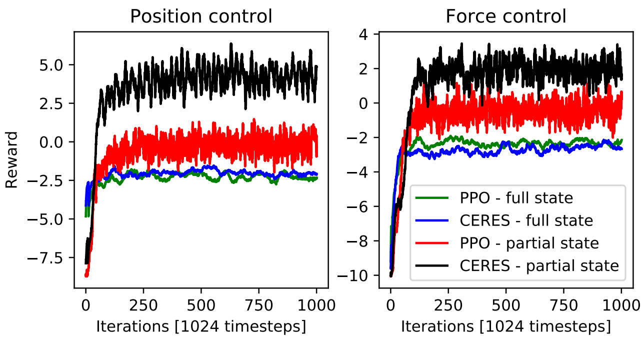


Figure 6: Reinforcement learning on random obstacles.

We consider four variants of this environment: two where the agent is controlled in position, with 2D position increments as actions, and two where it is controlled with 2D forces as actions, updating its velocity and position by consecutive integration. For each control setting, we consider the case where all observations are provided to the control policy, i.e., agent, target and obstacle informations, and the case where the control policy has only access to agent and target information, while the constraint network still has access to the whole state vector. We evaluate CERES against vanilla PPO, sharing the same reinforcement learning hyperparameters and random seeds and depict the resulting rewards in Fig. 6. Overall, while full-state tasks seem difficult to achieve in the first place, CERES enables the safe learning from fewer observations. Indeed, in such situations, considerations of distances can be left to the constraint network while the policy network can focus on general navigation.

6 Discussion and future work

In our work, we established that expert demonstrations could be used in a novel way, to learn safety constraints from positive and negative examples. When both are available, the resulting constraints can accelerate reinforcement learning by starting from and reaching higher rewards. Towards applications of practical interest, we derived a new algorithm, CERES, enabling the automatic discovery of such positive and negative examples, and thus the learning of safety constraints from scratch. On a task involving multi-step dynamics, we demonstrated that our approach could preserve such advantages in terms of rewards, while also enabling the main control policy to learn from fewer observations. Possible future developments include tackling real-world robotics applications and problems where success and failure metrics are more ambiguous.

References

Achiam, J.; Held, D.; Tamar, A.; and Abbeel, P. 2017. Constrained policy optimization. In *International Conference on Machine Learning*.

Cserna, B.; Doyle, W. J.; Ramsdell, J. S.; and Ruml, W. 2018. Avoiding dead ends in real-time heuristic search. In *AAAI*.

Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; and Wu, Y. 2017. Openai baselines. `https://github.com/openai/baselines`.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*.

Duan, Y.; Andrychowicz, M.; Stadie, B.; Ho, O. J.; Schneider, J.; Sutskever, I.; Abbeel, P.; and Zaremba, W. 2017. One-shot imitation learning. In *Advances in Neural Information Processing Systems*.

Eysenbach, B.; Gu, S.; Ibarz, J.; and Levine, S. 2018. Leave no trace: Learning to reset for safe and autonomous reinforcement learning. In *International Conference on Learning Representations*.

Gandhi, D.; Pinto, L.; and Gupta, A. 2017. Learning to fly by crashing. In *IEEE-RSJ International Conference on Intelligent Robots and Systems*.

Gao, Y.; Lin, J.; Yu, F.; Levine, S.; Darrell, T.; et al. 2018. Reinforcement learning from imperfect demonstrations. *arXiv preprint arXiv:1802.05313*.

Garcıa, J., and Fernández, F. 2015. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*.

Haarnoja, T.; Tang, H.; Abbeel, P.; and Levine, S. 2017. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*.

Haarnoja, T.; Pong, V.; Zhou, A.; Dalal, M.; Abbeel, P.; and Levine, S. 2018. Composable deep reinforcement learning for robotic manipulation. In *IEEE International Conference on Robotics and Automation*.

Hester, T.; Vecerik, M.; Pietquin, O.; Lanctot, M.; Schaul, T.; Piot, B.; Horgan, D.; Quan, J.; Sendonaris, A.; Dulac-Arnold, G.; et al. 2018. Deep q-learning from demonstrations. In *AAAI*.

Ho, J., and Ermon, S. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*.

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

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mattingley, J., and Boyd, S. 2012. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature*.

Ng, A. Y.; Russell, S. J.; et al. 2000. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*.

Peng, X. B.; Abbeel, P.; Levine, S.; and van de Panne, M. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics* 37(4).

Pham, T.-H.; De Magistris, G.; and Tachibana, R. 2018. OptLayer - Practical Constrained Optimization for Deep Reinforcement Learning in the Real World. In *IEEE International Conference on Robotics and Automation*.

Pinto, L.; Davidson, J.; Sukthankar, R.; and Gupta, A. 2017. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*.

Pomerleau, D. A. 1991. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation* 3(1).

Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; and Moritz, P. 2015. Trust region policy optimization. In *International Conference on Machine Learning*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of go with deep neural networks and tree search. *Nature*.

Su, J.; Vargas, D. V.; and Kouichi, S. 2017. One pixel attack for fooling deep neural networks. *arXiv preprint arXiv:1710.08864*.

Tobin, J.; Fong, R.; Ray, A.; Schneider, J.; Zaremba, W.; and Abbeel, P. 2017. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE-RSJ International Conference on Intelligent Robots and Systems*.

Wachi, A.; Sui, Y.; Yue, Y.; and Ono, M. 2018. Safe exploration and optimization of constrained mdps using gaussian processes. In *AAAI*.

Williams, G.; Wagener, N.; Goldfain, B.; Drews, P.; Rehg, J. M.; Boots, B.; and Theodorou, E. A. 2017. Information theoretic mpc for model-based reinforcement learning. In *IEEE International Conference on Robotics and Automation*.