Training Verifiably Robust Agents
Using Set-Based Reinforcement Learning

Manuel Wendl, Lukas Koller, Tobias Ladner, Matthias Althoff

Technical University of Munich, Germany

{manuel.wendl,lukas.koller,tobias.ladner,althoff}@tum.de

Abstract

Reinforcement learning often uses neural networks to solve complex control tasks. However, neural networks are sensitive to input perturbations, which makes their deployment in safety-critical environments challenging. This work lifts recent results from formally verifying neural networks against such disturbances to reinforcement learning in continuous state and action spaces using reachability analysis. While previous work mainly focuses on adversarial attacks for robust reinforcement learning, we train neural networks utilizing entire sets of perturbed inputs and maximize the worst-case reward. The obtained agents are verifiably more robust than agents obtained by related work, making them more applicable in safety-critical environments. This is demonstrated with an extensive empirical evaluation of four different benchmarks.

1 Introduction

In recent years, deep reinforcement learning using neural networks has significantly improved in solving complex control tasks (Mnih et al. 2015; Andrychowicz et al. 2020; Lillicrap et al. 2016). In many control tasks, state and action spaces are continuous, high-dimensional, and can be influenced by uncertainties such as sensor noises (Kober, Bagnell, and Peters 2013). However, such uncertainties present a severe challenge in reinforcement learning when storing policies as neural networks, which are sensitive to small perturbations in the input (Szegedy et al. 2014). This may lead to instabilities and safety violations of the controlled system: Fig. 1 (left) shows simulations of a navigation task where small input perturbations lead to trajectories that enter an unsafe set. Robust reinforcement learning algorithms aim to train controllers that are robust against input perturbations.

Related works on robust reinforcement learning (Zhang et al. 2021a; Huang et al. 2017; Deshpande, Minai, and Kumar 2021; Mandlekar et al. 2017; Lütjens, Everett, and How 2020; Zhang et al. 2021b) propose a competitive framework with an adversary (Moos et al. 2022; Pinto et al. 2017): In observation-robust algorithms, the adversary exploits the sensitivity of neural networks by choosing the worst-case observation for the current policy (Moos et al. 2022). The worst-case observation is often hard to compute (Madry et al. 2018); thus, different naive and gradient-based methods have been proposed to generate adversarial observations (Pattanaik et al. 2018; Mandlekar et al. 2017; Huang et al. 2017; Zhang et al. 2021a). For example, gradient-based methods often utilize the fast gradient sign method to approximate the worst input (Goodfellow, Shlens, and Szegedy 2015).

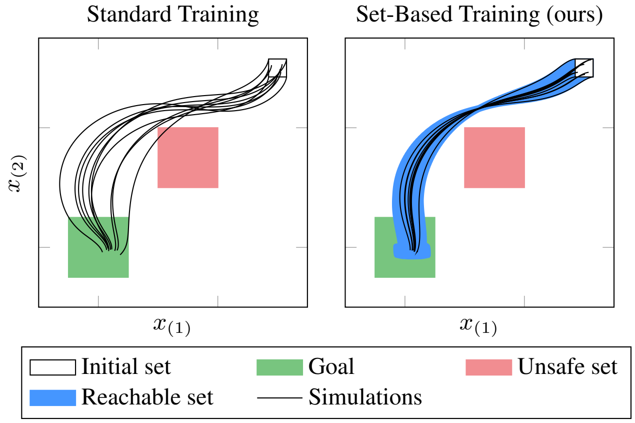

Figure 1: Comparison of standard and our novel set-based reinforcement learning on a navigation task. Left: Some trajectories of the standard agent intersect with the obstacle. Right: We can formally verify the safety of our robust agent.[1]

The obtained agents should be formally verified for safe deployment in safety-critical environments. Formal methods for neural networks have made significant progress in recent years (Manzanas Lopez et al. 2023; Brix et al. 2023), making it possible to verify entire neural network control systems: This is often achieved by (i) modeling the disturbed state of the system as a continuous set, (ii) computing the corresponding output set of the neural networks, and (iii) enclosing the evolution of the environment over time using reachability analysis. If the obtained reachable set does not violate any given specifications, the neural network control system is verified as shown in Fig. 1 (right).

The propagation of sets through neural networks can also be integrated into the training process to obtain neural networks that are robust against input uncertainties for classification tasks (Gowal et al. 2019; Koller, Ladner, and Althoff 2024). This is achieved by driving the entire output set to the target using a set-based loss, thereby training the accuracy

[1] Video: https://t1p.de/f8pqs

and robustness of the neural network. This work lifts these results to reinforcement learning. To summarize, our main contributions are:

- A novel set-based reinforcement learning algorithm based on the well-established deep deterministic policy gradient algorithm (Lillicrap et al. 2016) to train verifiable robust agents (Fig. 1).
- To do so, a novel set-based loss for regression tasks is developed using a rigorous analysis of the underlying set propagation.
- Our modular approach makes it possible to train all individual components in a set-based fashion.
- An extensive evaluation including a comparison with state-of-the-art adversarial training methods on four different control benchmarks.

2 Preliminaries

Notation

We write vectors as lowercase letters, matrices as uppercase letters, sets as calligraphic letters, and probability distributions as scriptfont letters. The set of natural numbers up to $n \in \mathbb{N}$ is written as $[n] = \{1, 2, \dots, n\} \subset \mathbb{N}$. A multidimensional interval is denoted by $\mathcal{I} = [l, u] \subseteq \mathbb{R}^n$, where $l \leq u$ holds elementwise for $l, u \in \mathbb{R}^n$. The gradient of a function f w.r.t. a variable x is denoted by $\nabla_x f(x, \cdot)$. The i-th element of a vector $v \in \mathbb{R}^n$ is written as $v_{(i)}$. The element in the i-th row and j-th column of a matrix is $M_{(i,j)}$, $M_{(i,\cdot)}$ refers to the i-th row, and $M_{(\cdot,j)}$ refers to the j-th column. The identity matrix is denoted by $I_n \in \mathbb{R}^{n \times n}$, and the vector that only contains ones or zeros is denoted by $\mathbf{1}$ or $\mathbf{0}$. The operator $\operatorname{diag} : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ returns a diagonal matrix with the vector elements on its diagonal; operation $\operatorname{sgn} : \mathbb{R}^{n \times m} \to \{-1, 0, 1\}^{n \times m}$ determines the sign of each matrix element. The horizontal concatenation of two matrices $A \in \mathbb{R}^{n \times m}$ and $B \in \mathbb{R}^{n \times p}$ is denoted by $[A\ B]$. We write the expected value of a random variable x under the condition $y \sim \mathscr{Y}$ as $\mathbb{E}_{y \sim \mathscr{Y}}[x(y)]$.

Neural Networks

A feed-forward neural network $N_\theta : \mathbb{R}^{n_0} \to \mathbb{R}^{n_\kappa}$ with learnable parameters θ consists of $\kappa \in \mathbb{N}$ alternating linear and activation layers, where the k-th layer has $n_k \in \mathbb{N}$ output neurons. The output $y = N_\theta(x) \in \mathbb{R}^{n_\kappa}$ is computed by propagating an input $x \in \mathbb{R}^{n_0}$ through all layers.

Definition 1 (Neural Network, (Bishop and Nasrabadi 2006, Sec. 5.1))**.** *Given a neural network N_θ and an input $x \in \mathbb{R}^{n_0}$, the output $y = N_\theta(x) \in \mathbb{R}^{n_\kappa}$ is computed by*

$$h_0 = x, \quad h_k = L_k(h_{k-1}) \quad \text{for } k \in [\kappa], \quad y = h_\kappa,$$

where

$$L_k(h_{k-1}) = \begin{cases} W_k\, h_{k-1} + b_k & \text{if } k\text{-th layer is linear,} \\ \sigma_k(h_{k-1}) & \text{otherwise,} \end{cases}$$

with weights $W_k \in \mathbb{R}^{n_k \times n_{k-1}}$, biases $b_k \in \mathbb{R}^{n_k}$, and elementwise activation functions $\sigma_k(\cdot)$.

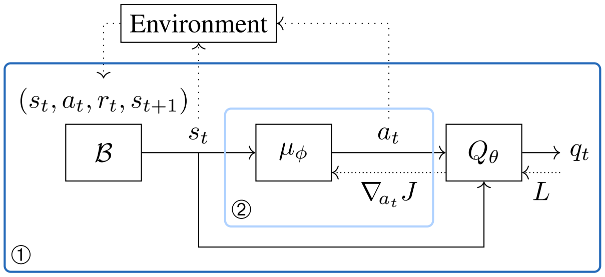

Figure 2: Illustration of the structure of the deep deterministic policy gradient algorithm; ① and ② show the components that are augmented through our set-based training (Sec. 3).

Deep Deterministic Policy Gradient

We focus on continuous control tasks with a multidimensional state space $\mathcal{S}$ and action space $\mathcal{A}$ (Januszewski et al. 2021; Recht 2019). Our set-based reinforcement learning approach is based on the deep deterministic policy gradient algorithm (Lillicrap et al. 2016), which consists of an actor $\mu_\phi \colon \mathcal{S} \to \mathcal{A}$ with parameters ϕ and a critic $Q_\theta \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ with parameters θ. At each time step t, the actor observes the current state of the system $s_t \in \mathcal{S}$ and returns an action $a_t = \mu_\phi(s_t) \in \mathcal{A}$, which controls the system until the next time step $t+1$. Using the reward function $r \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and the probabilistic state transition dynamics $p(s_{t+1} \mid s_t, a_t)$, the environment returns a reward r_t and its next state s_{t+1} (Fig. 2). These transitions (s_t, a_t, r_t, s_{t+1}) are stored in a buffer $\mathcal{B}$ for training (Fig. 2). The training objective is to find the policy that maximizes the discounted cumulative reward from the initial state s_0 (Silver et al. 2014, Eq. 8):

$$J(\mu_\phi) = \mathbb{E}_{s_t \sim \rho^\mu}\left[\sum_{i=0}^{\infty} \gamma^i\, r(s_t, \mu_\phi(s_t))\right], \tag{1}$$

with discount factor $\gamma \in [0, 1]$ and where ρ^μ denotes the discounted state visitation distribution for policy μ (Lillicrap et al. 2016, Sec. 2). The critic neural network Q_θ approximates the expected total discounted reward for action a_t in state s_t (Lillicrap et al. 2016, Eq. 3):

$$Q_\theta(s_t, a_t) = \mathbb{E}_{s_t \sim \rho^\mu}[r(s_t, a_t) + \gamma\, Q(s_{t+1}, \mu(s_{t+1}))]. \tag{2}$$

As the actor is deterministic, Q_θ can be learned off-policy by adding a random exploration noise to obtain a stochastic policy β (Lillicrap et al. 2016, Eq. 4):

$$L(\theta) = \mathbb{E}_{s_t \sim \rho^\beta, a_t \sim \beta}\left[(Q_\theta(s_t, a_t) - y_t)^2\right], \tag{3}$$

with targets (Lillicrap et al. 2016, Eq. 5)

$$y_t = r(s_t, a_t) + \gamma\, Q_\theta(s_{t+1}, \mu_\phi(s_{t+1})). \tag{4}$$

The actor is trained using the policy gradient (Lillicrap et al. 2016, Eq. 6):

$$\nabla_\phi J(\mu_\phi) \approx \mathbb{E}_{s_t \sim \rho^\beta}\left[\nabla_{a_t} Q_\theta(s_t, a_t)\big|_{a_t = \mu_\phi(s_t)} \nabla_\phi \mu_\phi(s_t)\right]. \tag{5}$$

Set-Based Computations

We model uncertainties using zonotopes due to their favorable computational complexity of the required operations:

Definition 2 (Zonotope (Girard 2005))**.** *Given a center $c \in \mathbb{R}^n$ and generators $G \in \mathbb{R}^{n\times q}$, a zonotope $\mathcal{Z}$ is defined as:*

$$\mathcal{Z} = \langle c, G \rangle_Z = \left\{ c + \sum_{j=1}^{q} \beta_j G_{(\cdot,j)} \,\middle|\, \beta_j \in [-1,1] \right\} \subset \mathbb{R}^n.$$

Given a zonotope $\mathcal{Z} \subset \mathbb{R}^n$, a matrix $A \in \mathbb{R}^{m\times n}$, and a vector $b \in \mathbb{R}^m$, the affine map is computed by (Althoff 2010, Sec. 2.4)

$$A\,\mathcal{Z} + b = \{Ax + b \,|\, x \in \mathcal{Z}\} = \langle A\,c + b, A\,G \rangle_Z. \quad (6)$$

The bounds $l, u \in \mathbb{R}^n$ of the enclosing interval of the zonotope, i.e., $\mathcal{Z} \subseteq [l, u]$, are computed by (Althoff 2010, Prop. 2.2)

$$l = c - |G|\,\mathbf{1}, \quad u = c + |G|\,\mathbf{1}, \quad (7)$$

and the diameter of the enclosing interval of a zonotope is given by

$$\mathrm{dia}(\mathcal{Z}) := u - l = 2\,|G|\,\mathbf{1}. \quad (8)$$

For some derivations, we write $\mathrm{lnDia}(G) := \ln(2\,|G|\mathbf{1})$ and $\mathrm{lnDia}'(G) := \nabla_G \,\mathrm{lnDia}(G) = \mathrm{diag}(|G|\mathbf{1})^{-1}\,\mathrm{sgn}(G)$ to make them more readable and concise.

The Minkowski sum of a zonotope $\mathcal{Z} = \langle c, G \rangle_Z$ and an interval $\mathcal{I} = [l, u]$ is computed by (Althoff 2010, Prop. 2.1 and Sec. 2.4):

$$\begin{aligned} \mathcal{Z} \oplus \mathcal{I} &= \{x_1 + x_2 \mid x_1 \in \mathcal{Z},\, x_2 \in \mathcal{I}\} \\ &= \langle c + 1/2(u + l), [G\ \mathrm{diag}(1/2(u - l))] \rangle_Z. \end{aligned} \quad (9)$$

Moreover, we calculate the Cartesian product of two zonotopes as in (Lützow and Althoff 2023, Sec. II.C).

Set Propagation through Neural Networks

Computing the exact output set $\mathcal{Y}^* = N_\theta(\mathcal{X})$ of a neural network for a given input set $\mathcal{X} \subset \mathbb{R}^{n_0}$ is computationally hard, i.e. for neural networks with ReLU-activation it is NP-hard (Katz et al. 2017). Thus, an enclosure of the output set $\mathcal{Y} \supseteq \mathcal{Y}^*$ is computed by conservatively propagating the input set through the layers of the neural network:

Proposition 1 (Neural Network Set Propagation (Singh et al. 2018))**.** *Given an input set $\mathcal{X}$, the output set of a neural network can be enclosed as:*

$$\begin{aligned} \mathcal{H}_0 &= \mathcal{X}, \\ \mathcal{H}_k^* \subseteq \mathcal{H}_k &= \texttt{enclose}(L_k, \mathcal{H}_{k-1}) \quad \textit{for } k \in [\kappa], \\ \mathcal{Y}^* \subseteq \mathcal{Y} &= \mathcal{H}_\kappa. \end{aligned}$$

The operation $\texttt{enclose}(L_k, \mathcal{H}_{k-1})$ encloses the output set of the k-th layer given the input set $\mathcal{H}_{k-1}$. If the layer L_k is linear, the affine map (6) is applied:

$$\texttt{enclose}(L_k, \mathcal{H}_{k-1}) = W_k\,\mathcal{H}_{k-1} + b_k. \quad (10)$$

Otherwise, the output set of an activation layer is enclosed by approximating its element-wise activation function with a linear function with slope $m_k \in \mathbb{R}^{n_k}$ and adding an appropriate approximation error $[\underline{d}, \overline{d}]$ for each dimension in the enclosing interval $[l_{\mathcal{H}_{k-1}}, u_{\mathcal{H}_{k-1}}] \supseteq \mathcal{H}_{k-1}$:

$$\texttt{enclose}(L_k, \mathcal{H}_{k-1}) = \mathrm{diag}(m_k)\,\mathcal{H}_{k-1} \oplus [\underline{d}, \overline{d}]. \quad (11)$$

Set-Based Training of Neural Networks

Set-based training augments the standard training of a neural network with set-based computing: We construct an input set $\mathcal{X}_i$ around each training sample x_i modeling the uncertainty, which is typically an ℓ_∞-ball around x_i. Ideally, the corresponding output set $\mathcal{Y}_i$ of the neural network is as small as possible around the target y_i. Therefore, a set-based loss combines a regular training loss with a norm of the output set to penalize its size (Koller, Ladner, and Althoff 2024). Thus, the backpropagation uses set-based computing to update the parameters of the neural networks.

Problem Statement

Given a reinforcement learning task with uncertain initial states $s_0 \in \mathcal{S}_0 \subset \mathbb{R}^n$, the training objective of an observation-robust agent μ_ϕ is to maximize the discounted cumulative rewards of the worst-case observation within the ℓ_∞-ball $\mathcal{S}_t = \langle s_t, \epsilon\, I_n \rangle_Z$ around the current state s_t :

$$\phi^* = \operatorname*{argmax}_{\phi}\, \mathbb{E}_{s_t\sim\rho^\mu}\left[\sum_{t=0}^{\infty} \gamma^t \min_{\tilde{s}_t \in \mathcal{S}_t} r(s_t, \mu_\phi(\tilde{s}_t))\right]. \quad (12)$$

3 Set-Based Reinforcement Learning

We lift standard actor-critic reinforcement learning to set-based reinforcement learning by using set-based training for the actor and the critic (SA-SC), which corresponds to a set-based evaluation of ① in Fig. 2. During training, we propagate entire sets through the actor and the critic, and apply a set-based loss to increase the robustness of the actor. To this end, we extend the critic loss (3), policy gradient (5), and the replay buffer to sets. The main steps are also provided in Alg. 1.

Given the current observation s_t of the actor, we add uncertainty by constructing the ℓ_∞-ball with perturbation radius $\epsilon \in \mathbb{R}_+$ around s_t:

$$\mathcal{S}_t = \langle s_t, \epsilon\, I \rangle_Z. \quad (13)$$

The set of actions of the actor is enclosed by propagating the set of observations $\mathcal{S}_t$ through the actor (Prop. 1):

$$\mathcal{A}_t = \langle c_{\mathcal{A}_t}, G_{\mathcal{A}_t} \rangle_Z = \texttt{enclose}(\mu_\phi, \mathcal{S}_t). \quad (14)$$

For the off-policy training of the critic, we perturb the set of actions with random exploration noise e_t (Lillicrap et al. 2016, Eq. 7): $\tilde{\mathcal{A}}_t := \mathcal{A}_t + e_t$, and compute the corresponding set of critic outputs:

$$\mathcal{Q}_t = \langle c_{\mathcal{Q}_t}, G_{\mathcal{Q}_t} \rangle_Z = \texttt{enclose}(Q_\theta, \mathcal{S}_t \times \tilde{\mathcal{A}}_t). \quad (15)$$

The environment receives the perturbed center of the action set $c_{\tilde{\mathcal{A}}_t} := c_{\mathcal{A}_t} + e_t$ and returns the reward $r(s_t, c_{\tilde{\mathcal{A}}_t})$ as well as the next state s_{t+1}, which are stored in the replay buffer $\mathcal{B}$ as transition $(s_t, \tilde{\mathcal{A}}_t, r_t, s_{t+1})$.

For the training of the critic neural network, we randomly sample n transitions from the buffer to get temporarily uncorrelated samples. For each transition $i \in [n]$, we compute the targets y_i using (4) and extend the set-based loss (Koller, Ladner, and Althoff 2024, Def. 4.1) to regression tasks:

Proposition 2 (Set-Based Regression Loss)**.** *Given an output set $\mathcal{Q}_i = \langle c_{\mathcal{Q}_i}, G_{\mathcal{Q}_i} \rangle_Z \subset \mathbb{R}$ and a target $y_i \in \mathbb{R}$, the set-based regression loss is defined as*

$$E_{Reg}(y_i, \mathcal{Q}_i) = \underbrace{\frac{1}{2}(c_{\mathcal{Q}_i} - y_i)^2}_{\textit{standard training loss}} + \underbrace{\frac{\eta_Q}{\epsilon}\,\ln\mathrm{Dia}(G_{\mathcal{Q}_i})}_{\textit{volume loss}},$$

with weighting factor $\eta_Q \in \mathbb{R}$ and perturbation radius $\epsilon \in \mathbb{R}_+$. The gradient of $E_{Reg}(y_i, \mathcal{Q}_i)$ w.r.t. $\mathcal{Q}_i$ is computed by:

$$\nabla_{\mathcal{Q}_i} E_{Reg}(y_i, \mathcal{Q}_i) = \left\langle c - y_i, \frac{\eta_Q}{\epsilon}\,\ln\mathrm{Dia}'(G_{\mathcal{Q}_i}) \right\rangle_Z.$$

Proof. See technical appendix. □

The derivation of the set-based loss function is deferred to Sec. 4. Intuitively, the set-based regression loss minimizes the half-squared error (Bishop and Nasrabadi 2006, Eq. 5.14) for the center of the output set and simultaneously improves the robustness by decreasing the volume of the output set. This is achieved by enforcing smaller absolute values in the generator matrix of the output sets.

The actor neural network is trained using a set-based policy gradient, which is also represented by a set:

Definition 3 (Set-Based Policy Gradient *SA-SC*)**.** *Given states $\mathcal{S}_i$ with the corresponding actions $\mathcal{A}_i = \langle c_{\mathcal{A}_i}, G_{\mathcal{A}_i} \rangle_Z$ and critic outputs $\mathcal{Q}_i = \langle c_{\mathcal{Q}_i}, G_{\mathcal{Q}_i} \rangle_Z$, a set-based policy gradient is defined as*

$$\nabla_{\mathcal{A}_i} J_{Set}(\mu_\phi) \coloneqq \left\langle \nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi), -\frac{\eta_\mu}{\epsilon}\, \nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi) \right\rangle_Z,$$

where

$$\begin{aligned} \nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi) &= \mathbb{E}_{s_i \sim \rho^\beta}\left[\nabla_{c_{\mathcal{A}_i}} c_{\mathcal{Q}_i}\right] \\ \textit{and } \nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi) &= \mathbb{E}_{s_i \sim \rho^\beta}\big[\omega\, \ln\mathrm{Dia}'(G_{\mathcal{A}_i}) \\ &\qquad + (1-\omega)\, \nabla_{G_{\mathcal{A}_i}} \ln\mathrm{Dia}'(G_{\mathcal{Q}_i})\big] \end{aligned}$$

with weighting factors $\eta_\mu \in \mathbb{R}_+$, $\omega \in [0, 1]$ and perturbation radius $\epsilon \in \mathbb{R}_+$.

The set-based policy gradient of the center $\nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi)$ corresponds to the standard deep deterministic policy gradient (5). For the set-based policy gradient of the generator matrix $\nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi)$, we deploy a weighted sum with factor ω consisting of two gradients to minimize the volume: The first term reduces the volume of the actions $\mathcal{A}_i$, and the second term reduces the volume of the critic outputs $\mathcal{Q}_i$, which we backpropagate through the critic to obtain the gradients in the space of the actions (Fig. 2). Finally, given the gradients w.r.t the output of the actor and the critic, we can update the respective parameters using set-based backpropagation (Koller, Ladner, and Althoff 2024, Prop. 14). Thereby, the set-based policy gradient simultaneously trains an accurate and robust actor.

Moreover, we can significantly speed up the training by setting $\omega = 1$, which corresponds to only using set-based training for the actor, while standard (point-based) training is used for the the critic (SA-PC). This corresponds to a set-based evaluation of ① in Fig. 2. Thus, the set propagation through the critic neural network can be omitted, and we do not need to store entire sets of actions $\mathcal{A}_i$ in the buffer $\mathcal{B}$. In this case, we can simplify the set-based policy gradient:

Definition 4 (Set-Based Policy Gradient *SA-PC*)**.** *For a set of states $\mathcal{S}_i$ with the corresponding set of actions $\mathcal{A}_i = \langle c_{\mathcal{A}_i}, G_{\mathcal{A}_i} \rangle_Z$, a set-based policy gradient is defined as*

$$\nabla_{\mathcal{A}_i} J_{Set}(\mu_\phi) \coloneqq \left\langle \nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi), -\frac{\eta_\mu}{\epsilon}\, \nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi) \right\rangle_Z,$$

where

$$\begin{aligned} \nabla_{c_{\mathcal{A}_i}} J_{Set}(\mu_\phi) &= \mathbb{E}_{s_i \sim \rho^\beta}\left[\nabla_{c_{\mathcal{A}_i}} Q_\theta(s_i, c_{\mathcal{A}_i})\right] \\ \textit{and } \nabla_{G_{\mathcal{A}_i}} J_{Set}(\mu_\phi) &= \mathbb{E}_{s_i \sim \rho^\beta}\left[\ln\mathrm{Dia}'(G_{\mathcal{A}_i})\right] \end{aligned}$$

with weighting factor $\eta_\mu \in \mathbb{R}_+$ and perturbation $\epsilon \in \mathbb{R}_+$.

Algorithm 1: Set-based reinforcement learning.

```
Randomly initialize $Q_\theta$, $\mu_\phi$ with $\theta$, $\phi$
Initialize replay buffer $\mathcal{B}$
for $episode = 1, \dots, maxEpisodes$ do
    Get initial observation $s_0$
    for $t = 0, 1, \dots, maxSteps$ do
                                                // Obtain new transition
        $\mathcal{S}_t \leftarrow \langle s_t, \epsilon I \rangle_Z$                  // perturb state (13)
        $\mathcal{A}_t \leftarrow$ enclose($\mu_\phi, \mathcal{S}_t$)   // evaluate actor (14)
        $\tilde{\mathcal{A}}_t \leftarrow \mathcal{A}_t + e_t$ with $e_t \sim \mathscr{E}$   // add noise (15)
        $r_t \leftarrow r(s_t, c_{\tilde{\mathcal{A}}_t})$                      // obtain reward
        $s_{t+1} \leftarrow$ execute action $c_{\tilde{\mathcal{A}}_t}$   // obtain next state
        Store transition $(s_t, \tilde{\mathcal{A}}_t, r_t, s_{t+1})$ in $\mathcal{B}$
                                                // Training step
        Sample batch of $n$ transitions from $\mathcal{B}$
        Compute target $y_i$ for each transition $i$    // (4)
        Update critic using regression loss   // Prop. 2
        Update actor using policy gradient    // Def. 3
```

4 Derivation of Set-Based Loss Functions

In this section, we motivate the choice of our set-based loss function (Prop. 2) and our set-based policy gradients (Def. 3 and 4). Please note that the training goal (12) is defined using probability theory, whereas our training algorithm (Alg. 1) only uses set-based computing. To this end, we connect set-based computing and probability theory by assuming a probability distribution over the considered sets. While maximizing probabilities is a standard procedure to derive loss functions (Bishop and Nasrabadi 2006, Sec. 1.2.5), lifting it to set-based computing has the unique advantage of integrating formal methods into the training process. For our derivations, we make use of a conditional posterior distribution, which can be rewritten to be proportional to a likelihood function and a prior distribution (Bishop and Nasrabadi 2006, Eq. 1.44):

$$\text{cond. posterior} \propto \text{likelihood} \cdot \text{prior}. \tag{16}$$

This reformulation is used as the posterior and is not directly obtainable, but we can assume distributions for the likelihood and the prior to get an estimate. In our case, the likelihood corresponds to the standard (point-based) training goal, and the prior penalizes the volume of the computed sets. As these sets are represented by zonotopes and thus point-symmetric, we additionally assume that the expected value over a zonotope is its center:

$$\mathbb{E}_{z\sim\mathcal{Z}}[z] = c. \tag{17}$$

Set-Based Regression Loss

We sample random transitions $i \in [n]$ from the buffer $\mathcal{B}$ to obtain a state s_i and the corresponding actions $\tilde{a}_i \sim \tilde{\mathcal{A}}_i$. For each transition, we use (15) to obtain the critic output $\mathcal{Q}_i$ and use (4) to obtain the target y_i for each critic output $q_i \sim \mathcal{Q}_i$ using the rewards and next states stored in the buffer.

To train the critic Q_θ, we want to maximize the probability $p_\theta(q_i|y_i, s_i, \tilde{a}_i, \beta^{-1})$. Unfortunately, this probability can not be computed directly. Hence, we model this probability as a conditional posterior using (16):

$$\underbrace{p_\theta(q_i|y_i, s_i, \tilde{a}_i, \beta^{-1})}_{\text{cond. posterior}} \propto \underbrace{p(y_i|q_i, \beta^{-1})}_{\text{likelihood}} \underbrace{p_\theta(q_i|s_i, \tilde{a}_i)}_{\text{prior}}. \tag{18}$$

For the prior, we assume that q_i is uniformly distributed over the interval $[l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}] \supseteq \mathcal{Q}_i \subset \mathbb{R}$, as Prop. 1 is also defined over the enclosing interval. Thus, the prior is given by

$$p_\theta(q_i|s_i, \tilde{a}_i) = \mathscr{U}(q_i|l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}) = \operatorname{dia}(\mathcal{Q}_i)^{-1}. \tag{19}$$

As the critic learns a regression task, we assume that the targets y_i are normally distributed with mean q_i and variance β^{-1} to obtain the likelihood function (Bishop and Nasrabadi 2006, Eq. 1.60):

$$\begin{aligned} p(y_i|q_i, \beta^{-1}) &= \mathscr{N}(y_i|q_i, \beta^{-1}) \\ &= \sqrt{\beta/2\pi}\, \exp\bigl(-\beta/2\, (q_i - y_i)^2\bigr). \end{aligned} \tag{20}$$

Since we observe not a single q_i but an entire set $\mathcal{Q}_i$, we use the expected value $\mathbb{E}_{q_i\sim\mathcal{Q}_i}[q_i] = c_{\mathcal{Q}_i}$ (17) in the likelihood function (Bishop and Nasrabadi 2006, Sec. 10.3). Thus, we obtain the following term to be maximized:

$$p_\theta(q_i|y_i, s_i, \tilde{a}_i, \beta^{-1}) \propto p(y_i|c_{\mathcal{Q}_i}, \beta^{-1})\, p_\theta(q_i|s_i, \tilde{a}_i) \tag{21}$$

Finally, we apply the negative logarithm and set $\beta = \left(\eta_Q/\epsilon\right)^{-1}$ to obtain our set-based loss (Prop. 2):

$$\begin{aligned} &-\ln\bigl(p(y_i|c_{\mathcal{Q}_i}, \beta^{-1})\, p_\theta(q_i|s_i, \tilde{a}_i)\bigr) \\ &\overset{(20),\,(19)}{=} -\ln\bigl(\mathscr{N}(y_i|q_i, \beta^{-1})\ \operatorname{dia}(\mathcal{Q}_i)^{-1}\bigr) \\ &\propto \beta/2\, (c_{\mathcal{Q}_i} - y_i)^2 + \operatorname{lnDia}(G_{\mathcal{Q}_i}) \\ &\overset{\text{Prop. 2}}{\propto} E_{Reg}(y_i, \mathcal{Q}_i). \end{aligned} \tag{22}$$

We choose β that way for easier fine-tuning (Koller, Ladner, and Althoff 2024, Def. 7).

Set-Based Policy Gradient

For a state s_i sampled from the buffer $\mathcal{B}$, we derive the set-based policy gradient analogous to (Xiao and Wang 2022), to maximize the probability of an action $a_i \sim \mathcal{A}_i$ being optimal given s_i – which is again not directly obtainable. Thus, we introduce a binary variable o_i indicating whether a_i is optimal and abbreviate $o_i = 1$ by o_i (Xiao and Wang 2022, Sec. 3.1).

We again model this probability as a conditional posterior over the action output using (16):

$$\underbrace{p_{\phi,\theta}(a_i|o_i, s_i, q_i, \alpha)}_{\text{cond. posterior}} \propto \underbrace{p(o_i|q_i, \alpha)}_{\text{likelihood}} \underbrace{p_{\phi,\theta}(a_i|q_i, s_i)}_{\text{prior}}. \tag{23}$$

The likelihood is assumed to be exponentially distributed with parameter $\alpha \in \mathbb{R}_+$ (Xiao and Wang 2022, Sec. 3.2):

$$p(o_i|a_i, q_i, \alpha) = \exp(\alpha^{-1}\, q_i). \tag{24}$$

The prior is also modeled as a conditional posterior (16):

$$\underbrace{p_{\phi,\theta}(a_i|q_i, s_i)}_{\text{cond. posterior}} = \underbrace{p_\theta(q_i|a_i, s_i)}_{\text{likelihood}} \underbrace{p_\phi(a_i|s_i)}_{\text{prior}}, \tag{25}$$

where the likelihood function of q_i and the prior for a_i are uniform distributions over the enclosing intervals $[l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}] \supseteq \mathcal{Q}_i \subset \mathbb{R}$ and $[l_{\mathcal{A}_i}, u_{\mathcal{A}_i}] \supseteq \mathcal{A}_i \subset \mathbb{R}^{n_{\mathcal{A}_i}}$ analogous to (19):

$$\begin{aligned} p_\theta(q_i|a_i, s_i) &= \mathscr{U}(q_i|l_{\mathcal{Q}_i}, u_{\mathcal{Q}_i}) = \operatorname{dia}(\mathcal{Q}_i)^{-1}, \\ p_\phi(a_i|s_i) &= \mathscr{U}(a_i|l_{\mathcal{A}_i}, u_{\mathcal{A}_i}) = \prod_{j=1}^{n_{\mathcal{A}_i}} \operatorname{dia}(\mathcal{A}_i)_{(j)}^{-1}. \end{aligned} \tag{26}$$

Please note that these two probabilities correspond to the evaluation of the actor and the critic, respectively. For the likelihood function of (23), the expected value $\mathbb{E}_{q_i\sim\mathcal{Q}_i}[q_i] = c_{\mathcal{Q}_i}$ (17) is used, and taking the logarithm obtains us:

$$\begin{aligned} &\ln(p(o_i|c_{\mathcal{Q}_i}, \alpha)\, p_\phi(a_i|s_i)\, p_\theta(q_i|a_i, s_i)) \\ &\overset{(24),\,(26)}{=} \ln(\exp(\alpha^{-1}\, c_{\mathcal{Q}_i})) \\ &\quad + \ln\left(\prod_{j=1}^{n_\kappa} \operatorname{dia}(\mathcal{A}_i)_{(j)}^{-1}\right) + \ln\bigl(\operatorname{dia}(\mathcal{Q}_i)^{-1}\bigr) \\ &= \alpha^{-1} c_{\mathcal{Q}_i} - \mathbf{1}^\top \operatorname{lnDia}(G_{\mathcal{A}_i}) - \operatorname{lnDia}(G_{\mathcal{Q}_i}) \end{aligned} \tag{27}$$

The set-based policy gradient for SA-SC (Def. 3) is derived by differentiation, where we again set the weighting factor $\alpha = \left(\eta_\mu/\epsilon\right)^{-1}$ for easier fine-tuning (Koller, Ladner, and Althoff 2024, Def. 7). Moreover, we introduced a factor $\omega \in [0, 1]$ to weight the gradients of the prior terms of $\mathcal{A}_i$ and $\mathcal{Q}_i$.

Derivation of SA-PC For *SA-PC* only the actor is trained using set-based training, while the critic uses standard (point-based) training (① in Fig. 2). Thus, we can drop the prior for the critic output q_i as this is no longer evaluated set-based and use the expected value $\mathbb{E}_{a_i\sim\mathcal{A}_i}[a_i] = c_{\mathcal{A}_i}$ for the likelihood:

$$\underbrace{p(a_i|o_i, s_i, \alpha, \phi)}_{\text{cond. posterior}} \propto \underbrace{p(o_i|s_i, c_{\mathcal{A}_i}, \alpha)}_{\text{likelihood}} \underbrace{p_\phi(a_i|s_i)}_{\text{prior}}. \tag{28}$$

Applying the logarithm while keeping our assumption on the likelihood function and the prior obtains us:

$$\begin{aligned} &\ln(p(o_i|s_i, c_{\mathcal{A}_i}, \alpha)\, p_\phi(a_i|s_i)) \\ &\overset{(24),\,(26)}{=} \alpha^{-1} Q_\theta(s_i, c_{\mathcal{A}_i}) + \ln\left(\prod_{j=1}^{n_{\mathcal{A}_i}} \operatorname{dia}(\mathcal{A}_i)_{(j)}^{-1}\right) \\ &= \alpha^{-1} Q_\theta(s_i, c_{\mathcal{A}_i}) - \mathbf{1}^\top \operatorname{lnDia}(G_{\mathcal{A}_i}). \end{aligned} \tag{29}$$

The set-based policy gradient for SA-PC (Def. 4) is derived by differentiation and choosing α as above. This also corresponds to setting $\omega = 1$ in Def. 3.

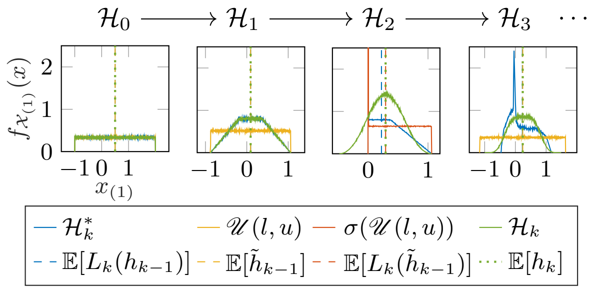

Figure 3: Probability density function of a zonotope propagated through a neural network with ReLU-activations: Exact density function obtained via sampling (blue), interval enclosure (yellow), and the density of sets obtained using Prop. 1 with uniformly distributed $\beta_j \sim \mathscr{U}(-1,1)$ (Def. 2) (green).

Expectation Preserving Image Enclosure

For (21), (27) and (29), we simplify the likelihood with the expected value of the neural network output, i.e., the center. This simplification is justified with Prop. 3. In Fig. 3, we plot the probability distributions of a set propagation and visualize the expected value for the first neuron of each layer. The expected value is trivially preserved for linear layers as they are computed exactly. For nonlinear layers, we observe that the expected value of the interval enclosure (red vertical line in third plot) is preserved through the enclosure (green vertical line; Prop. 1), with only small deviations to the true expected value. Formally, this is stated in the following proposition:

Proposition 3 (Tight Expectation-Preserving Set Propagation)**.** *Given a neural network and an input set $\mathcal{H}_{k-1}$ with the enclosing interval $[l_{k-1}, u_{k-1}] \supseteq \mathcal{H}_{k-1}$, the expected value of the enclosure of the output set of the k-th layer is*

$$\mathbb{E}_{h_k \sim \mathcal{H}_k}[h_k] = \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[L_k(h_{k-1})],$$

with $\mathcal{H}_k = \texttt{enclose}(L_k, \mathcal{H}_{k-1})$. Moreover, for neural networks with only ReLU*-activations, the approximation errors are minimal with approximation slope $m_k = (\sigma_k(u_{k-1}) - \sigma_k(l_{k-1}))/(u_{k-1} - l_{k-1})$.*

Proof. See technical appendix. □

5 Evaluation

We use the MATLAB toolbox CORA (Althoff 2015) to implement our novel set-based reinforcement learning algorithm and compare the *SA-PC* and the *SA-SC* implementation against standard (point-based) training (*PA-PC*) and two state-of-the-art adversarial methods: *Naive*- and *Grad*-based implementations from (Pattanaik et al. 2018, Alg. 2 and 4), which compute adversarial attacks to approximate the worst-case observation within a perturbation set (12).

For our evaluation, we use reachability analysis using CORA to compute all reachable states until a specified time horizon $t_{\text{end}} \in \mathbb{R}_+$, enclosing all states within the time interval $[0, t_{\text{end}}]$. We add uncertainties (13) at each time step $t \in [t_{\text{end}}]$ and consider all uncertainties from $[0,t]$ to formally verify the behavior of an agent over time (Fig. 1). As this process is outer-approximative, we can compare the different training approaches using a lower bound of the worst-case return of the set of states $\mathcal{S}_t = \langle c_t, G_t \rangle_Z$ obtained by CORA. For reward functions of the form $r(s_t, a_t) = w^\top |s_t - s^*|$, we can use set-based computing to obtain this lower bound:

$$\begin{aligned} \underline{V}_\mu(s_0) &= \sum_{t=0}^{t_{\text{end}}} \gamma^t \max_{s_t \in \mathcal{S}_t} w^\top |s_t - s^*| \\ &\overset{(6),(8)}{=} \sum_{t=0}^{t_{\text{end}}} \gamma^t \big(w^\top |c_t - s^*| + \operatorname{dia}(w^\top \mathcal{S}_t)/2\big). \end{aligned} \tag{30}$$

The evaluation is done with three different benchmarks and an additional one included in the technical appendix:

1D Quadrocopter The state is $s = [z \quad \dot{z}]^\top$, with altitude z, vertical velocity $\dot{z}$, and dynamics (Yuan et al. 2022):

$$\dot{s} = \begin{bmatrix} \dot{z} & \frac{a+1}{2\,m} - g \end{bmatrix}^\top, \tag{31}$$

with action space $a \in [-1, 1]$, gravity $g = 9.81$, and mass $m = 0.05$. Starting from initial states $s_0 \in [[-4 \quad 0]^\top, [4 \quad 0]^\top]$, the quadrocopter is stabilized at $s^* = \mathbf{0}$; the reward function is $r(s_t, a_t) = -[1 \quad 0.01]|s_{t+1} - s^*|$.

Inverted Pendulum The state is $s = \begin{bmatrix} \theta & \dot{\theta} \end{bmatrix}^\top$, with angle θ, angular velocity $\dot{\theta}$, and dynamics (Krasowski et al. 2023):

$$\dot{s} = \begin{bmatrix} \dot{\theta} & \frac{g}{l}\sin(\theta) + \frac{1}{m\,l^2}\,a \end{bmatrix}^\top, \tag{32}$$

with action space $a \in [-15, 15]$, gravity $g = 9.81$, mass $m = 1$, and length $l = 1$. The goal to stabilize the pendulum in the upright position $s^* = \mathbf{0}$; the reward function is $r(s_t, a_t) = -[1 \quad 0.01]\,|(s_{t+1} - s^*)|$.

Navigation Task We use a unicycle model with states $s = [x \quad y \quad \theta \quad v]^\top$ and dynamics (Lopez et al. 2022):

$$\dot{s} = \begin{bmatrix} v\,\cos(\theta) & v\,\sin(\theta) & a_{(1)} & a_{(2)} \end{bmatrix}^\top, \tag{33}$$

with action space $a \in [-\mathbf{1}, \mathbf{1}]$. From starting point $s_0 = [3 \quad 3 \quad 0 \quad 0]^\top$, the task is to navigate to the goal $s^* = \mathbf{0}$ without colliding with an obstacle $\mathcal{O} = [\mathbf{1}, 2 \cdot \mathbf{1}]$. The reward function is $r(s_t, a_t) = -[1 \quad 1 \quad 0 \quad 0]\,|s_{t+1} - s^*| - c$, with $c = 1$ if $s_{t+1} \in \mathcal{O}$ and otherwise $c = 0$.

Every benchmark is trained with 5 different random seeds. We use neural networks with ReLU activations and two hidden layers of 64 and 32 neurons for the actor and critic networks. The output layer of the actor has an additional hyperbolic tangent activation to match the action space. All other hyperparameters are listed in the technical appendix.

We compare the performance of the obtained agents for an increasing perturbation ϵ_{test}:

- With increasing ϵ_{test}, the algorithms *SA-PC* and *SA-SC* show a better $\underline{V}_\mu(s_0)$ on all benchmarks compared to the *PA-PC*, *Naive* and *Grad* algorithms (Fig. 4).
- The *SA-PC* implementation trains more conservative actors, which perform best for large ϵ_{test} while having a worse $\underline{V}_\mu(s_0)$ for small ϵ_{test} compared to all other implementations. The conservative *SA-PC* actor for the *1D Quadrocopter* is the most robust but reaches the goal later using lower vertical velocities (Fig. 5).

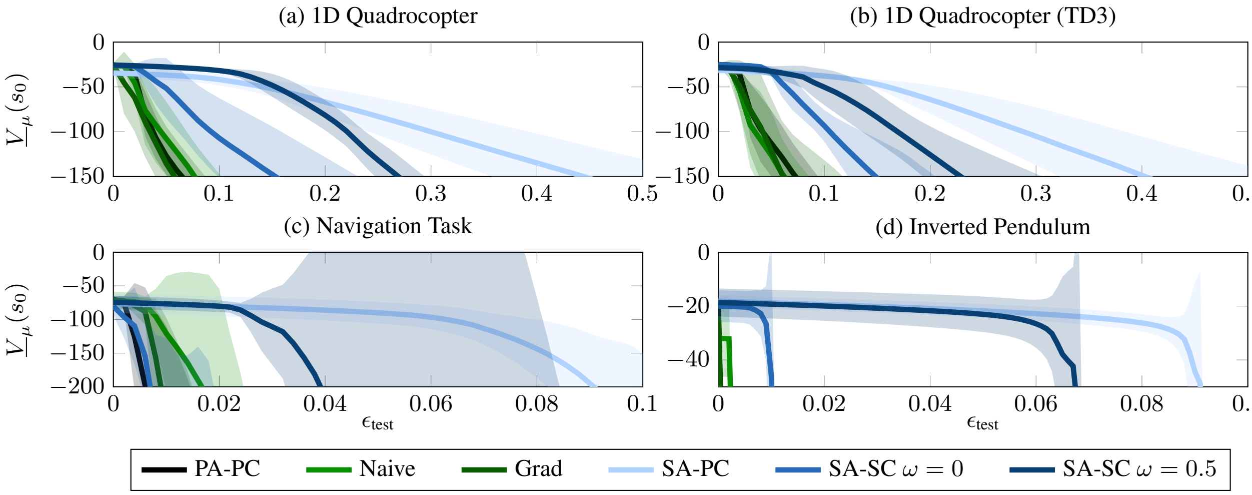

Figure 4: Comparison of $\underline{V}_\mu(s_0)$ for the (a) 1D Quadrocopter, (c) Navigation Task, and (d) Inverted Pendulum benchmark. The TD3 implementation is compared in (b) for the 1D Quadrocopter.

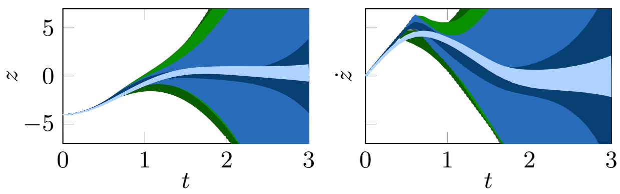

Figure 5: Quad. 1D: Comparison of the reachable altitudes z and vertical speeds $\dot{z}$ for $\epsilon_{\text{test}} = 0.15$.

- For the *SA-SC* implementations, $\underline{V}_\mu(s_0)$ is comparable to *PA-PC* for small ϵ_{test}. For increasing ϵ_{test}, *SA-SC* with $\omega = 0.5$ shows similar behavior as *SA-PC*. For *SA-SC* with $\omega = 0$, $\underline{V}_\mu(s_0)$ decreases faster (Fig. 4).
- The $\underline{V}_\mu(s_0)$ of the *Grad* and *Naive* methods are similar to *PA-PC*. This may relate to the strength of the proposed attacks in *Grad* and *Naive* training (Pattanaik et al. 2018), which are strongly dependent on the quality of the trained critic (Zhang et al. 2021a, Sec. 3.5). For the *1D Quadrocopter*, Fig. 6 compares $\underline{V}_\mu(s_0)$ with the return under the adversarial attacks of the *Naive* and *Grad* algorithms. The *Naive* and *Grad* training methods perform best for the respective attacks but not for entire noise sets.
- Our proposed set-based reinforcement learning can be directly extended for the Twin Delayed Deep Deterministic policy gradient algorithm (TD3) (Fujimoto, van Hoof, and Meger 2018) and other ensemble algorithms (Januszewski et al. 2021). Fig. 4b shows a similar performance of the set-based TD3 implementation for the *1D Quadrocopter*.

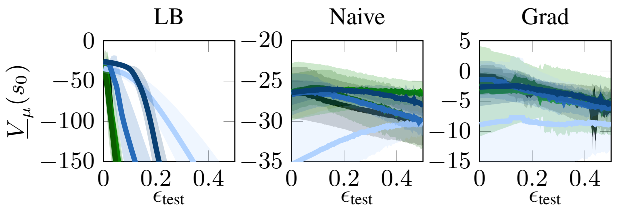

Figure 6: Quad. 1D: Comparison of the return under *Naive* and *Grad* attacks with $\underline{V}_\mu(s_0)$.

Table 1: Training times [s/10 epochs]

Benchmark	PA-PC	Naive	Grad	SA-PC	SA-SC
1D Quad.	1.58	2.10	1.98	2.77	7.92
Pendulum	1.82	2.04	2.02	2.87	7.66
Nav. Task	2.35	2.89	2.84	4.35	12.43

Set-based reinforcement learning can be efficiently computed batch-wise on a GPU, but the memory load remains challenging. Especially for *SA-SC*, storing entire action sets in $\mathcal{B}$ is memory-consuming. The large number of generators slows down the computations. The different runtimes for 10 learning epochs are listed in Tab. 1 and were run on a server with two *AMD EPYC 7763* 64 core processors, 2 TB RAM, and an *NVIDIA A100-PCIE* 40 GB GPU.

6 Conclusion

To the best of our knowledge, we introduce the first set-based observation-robust reinforcement learning algorithm. Unlike other algorithms that rely on adversarial inputs, our approach is unique in its use of set-based neural network training, working with entire sets of inputs and gradients. The set-based losses are motivated by a rigorous analysis of the underlying set-propagation. Our experimental results on different benchmarks demonstrate the efficacy of set-based reinforcement learning. Particularly, set-based reinforcement learning allows the formal verification of trained controllers even for large perturbation sets, which is essential for their deployment in safety-critical environments. Consequently, set-based reinforcement learning is an effective, novel approach for training robust neural network controllers.

Acknowledgments

This work was partially supported by the project SPP 2422 (No. 500936349), the project FAI (No. 286525601), and the projcet SFB 1608 (No. 501798263), all funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation).

References

Althoff, M. 2010. *Reachability analysis and its application to the safety assessment of autonomous cars*. Ph.D. thesis, Technical University of Munich.

Althoff, M. 2015. An introduction to CORA 2015. In *Proc. of the workshop on applied verification for continuous and hybrid systems*, 120–151.

Andrychowicz, O. M.; Baker, B.; Chociej, M.; Jozefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; et al. 2020. Learning dexterous in-hand manipulation. *The Int. Journal of Robotics Research*, 39: 3–20.

Au, C.; and Tam, J. 1999. Transforming Variables Using the Dirac Generalized Function. *The American Statistician*, 53(3): 270–272.

Bishop, C. M.; and Nasrabadi, N. M. 2006. *Pattern recognition and machine learning*, volume 4. Springer.

Brix, C.; Bak, S.; Liu, C.; and Johnson, T. T. 2023. The Fourth Int. Verification of Neural Networks Competition (VNN-COMP 2023): Summary and Results. arXiv:2312.16760.

Deshpande, A. M.; Minai, A. A.; and Kumar, M. 2021. Robust deep reinforcement learning for quadcopter control. *Int. Federation of Automatic Control (IFAC-PapersOnLine)*, 54(20): 90–95.

Fujimoto, S.; van Hoof, H.; and Meger, D. 2018. Addressing Function Approximation Error in Actor-Critic Methods. In *Proc. of the Int. Conf. on Machine Learning (ICML)*.

Girard, A. 2005. Reachability of uncertain linear systems using zonotopes. In *Int. Workshop on Hybrid Systems: Computation and Control*, 291–305.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and Harnessing Adversarial Examples. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

Gowal, S.; Dvijotham, K.; Stanforth, R.; Bunel, R.; Qin, C.; Uesato, J.; Arandjelovic, R.; Mann, T. A.; and Kohli, P. 2019. Scalable Verified Training for Provably Robust Image Classification. In *Proc. of the IEEE/CVF Int. Conf. on Computer Vision (ICCV)*, 4841–4850.

Hinton, G. E.; and Ghahramani, Z. 1997. Generative models for discovering sparse distributed representations. *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, 352(1358): 1177–1190.

Huang, S. H.; Papernot, N.; Goodfellow, I. J.; Duan, Y.; and Abbeel, P. 2017. Adversarial Attacks on Neural Network Policies. *CoRR*.

Januszewski, P.; Olko, M.; Królikowski, M.; Swiatkowski, J.; Andrychowicz, M.; Kuciński, Ł.; and Miłoś, P. 2021. Continuous Control With Ensemble Deep Deterministic Policy Gradients. In *Deep RL Workshop NeurIPS 2021*.

Katz, G.; Barrett, C.; Dill, D. L.; Julian, K.; and Kochenderfer, M. J. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *Computer Aided Verification (CAV)*, 97–117.

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

Koller, L.; Ladner, T.; and Althoff, M. 2024. Set-Based Training for Neural Network Verification. arXiv:2401.14961.

Krasowski, H.; Thumm, J.; Müller, M.; Schäfer, L.; Wang, X.; and Althoff, M. 2023. Provably Safe Reinforcement Learning: Conceptual Analysis, Survey, and Benchmarking. *Transactions on Machine Learning Research*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

Lopez, D. M.; Althoff, M.; Benet, L.; Chen, X.; Fan, J.; Forets, M.; Huang, C.; Johnson, T. T.; Ladner, T.; Li, W.; Schilling, C.; and Zhu, Q. 2022. ARCH-COMP22 Category Report: Continuous and Hybrid Systems with Linear Continuous Dynamics. In *Proc. of 9th Int. Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, volume 90, 142–184. EasyChair.

Lütjens, B.; Everett, M.; and How, J. P. 2020. Certified adversarial robustness for deep reinforcement learning. In *Proc. of the Conf. on Robot Learning (CoRL)*, 1328–1337.

Lützow, L.; and Althoff, M. 2023. Reachability analysis of ARMAX models. In *Proc. of the IEEE Conf. on Decision and Control (CDC)*, 7027–7034.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

Mandlekar, A.; Zhu, Y.; Garg, A.; Fei-Fei, L.; and Savarese, S. 2017. Adversarially robust policy learning: Active construction of physically-plausible perturbations. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 3932–3939.

Manzanas Lopez, D.; Althoff, M.; Forets, M.; Johnson, T. T.; Ladner, T.; and Schilling, C. 2023. ARCH-COMP23 category report: Artificial intelligence and neural network control systems (AINNCS) for continuous and hybrid systems plants. In *Proc. of Int. Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH23)*.

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*, 518(7540): 529–533.

Moos, J.; Hansel, K.; Abdulsamad, H.; Stark, S.; Clever, D.; and Peters, J. 2022. Robust reinforcement learning: A review of foundations and recent advances. *Machine Learning and Knowledge Extraction*, 4(1): 276–315.

Pattanaik, A.; Tang, Z.; Liu, S.; Bommannan, G.; and Chowdhary, G. 2018. Robust Deep Reinforcement Learning with Adversarial Attacks. In *Proc. of the Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*, 2040–2042.

Pinto, L.; Davidson, J.; Sukthankar, R.; and Gupta, A. 2017. Robust adversarial reinforcement learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2817–2826.

Recht, B. 2019. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2: 253–279.

Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; and Riedmiller, M. 2014. Deterministic Policy Gradient Algorithms. In Xing, E. P.; and Jebara, T., eds., *Proc. of the Int. Conf. on Machine Learning (ICML)*, volume 32, 387–395.

Singh, G.; Gehr, T.; Mirman, M.; Püschel, M.; and Vechev, M. 2018. Fast and Effective Robustness Certification. In Bengio, S.; Wallach, H.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Socci, N.; Lee, D.; and Seung, H. S. 1997. The rectified Gaussian distribution. *Advances in neural information processing systems*, 10.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I. J.; and Fergus, R. 2014. Intriguing properties of neural networks. In Bengio, Y.; and LeCun, Y., eds., *Proc. of the Int. Conf. on Learning Representations (ICLR)*.

Xiao, T.; and Wang, S. 2022. Towards Off-Policy Learning for Ranking Policies with Logged Feedback. *Proc. of the AAAI Conf. on Artificial Intelligence*, 36(8): 8700–8707.

Yuan, Z.; Hall, A. W.; Zhou, S.; Brunke, L.; Greeff, M.; Panerati, J.; and Schoellig, A. P. 2022. Safe-Control-Gym: A Unified Benchmark Suite for Safe Learning-Based Control and Reinforcement Learning in Robotics. *IEEE Robotics and Automation Letters*, 7: 11142–11149.

Zhang, H.; Chen, H.; Boning, D.; and Hsieh, C. J. 2021a. Robust Deep Reinforcement Learning against Adversarial Perturbations on State Observations. In *Int. Conf. on Learning Representations (ICLR)*.

Zhang, H.; Chen, H.; Boning, D.; and Hsieh, C.-J. 2021b. Robust reinforcement learning on state observations with learned optimal adversary. *arXiv preprint arXiv:2101.08452*.

Table 2: Training parameters for *PA-PC*, *SA-PC* and *SA-SC*.

Parameter	DDPG	TD3
Actor learning rate	$1 \cdot 10^{-4}$	$1 \cdot 10^{-4}$
Critic learning rate	$1 \cdot 10^{-3}$	$1 \cdot 10^{-3}$
Critic L_2 weight regularization λ_Q	0.01	0
Discount factor γ	0.99	0.99
Target update factor τ	0.05	0.05
Exploration noise std. deviation σ	0.1	0.1, 0.2
Batchsize	64	64
Buffersize	$1 \cdot 10^{6}$	$1 \cdot 10^{6}$
Episodes	2000	2000
Perturbation radius ϵ	0.1	0.1
Actor weighting factor η_μ	0.1	0.1
Critic weighting factor η_Q	0.01	0.01

A Evaluation Details

Hyperparameters

Tab. 2 show the training hyperparameters.

Additional Benchmark

2D Quadrocopter The state of the system is defined as $s = \begin{bmatrix} x & \dot{x} & z & \dot{z} & \theta & \dot{\theta} \end{bmatrix}^\top$, with horizontal displacement x, horizontal velocity $\dot{x}$, altitude z, vertical velocity $\dot{z}$, angle θ, angular velocity $\dot{\theta}$ and dynamics (Yuan et al. 2022):

$$\dot{s} = \begin{bmatrix} \dot{x} \\ \sin(\theta)\frac{\tilde{a}_{(1)}+\tilde{a}_{(2)}}{m} \\ \dot{z} \\ \cos(\theta)\frac{\tilde{a}_{(1)}+\tilde{a}_{(2)}}{m} - g \\ \dot{\theta} \\ \frac{l(\tilde{a}_{(2)}-\tilde{a}_{(1)})}{\sqrt{2}J_y} \end{bmatrix}, \qquad (34)$$

where $\tilde{a} = (1 + \frac{1}{2}\, a)\frac{m\, g}{2}$ with action $a \in [-\mathbf{1}, \mathbf{1}] \subset \mathbb{R}^2$. The constant $g = 9.81$ defines gravity, $m = 0.027$ is the mass of the quadrocopter, $l = 0.0397$ and $J_y = 1.4 \cdot 10^{-4}$ defines the arm length of the propeller mount and the moment of inertia around the y axis. The reward function is given by $r(s_t, a_t) = -[1 \quad 0.01 \quad 1 \quad 0.01 \quad 0 \quad 0]|s_{t+1} - s^*|$, with the goal to stabilize the quadrocopter at $s^* = \mathbf{0}$. Fig. 7 compares the lower bounds $\underline{V}_\mu(s_0)$ for the different training algorithms.

B Proofs of Sec. 3

Proposition 2. *Given an output set $\mathcal{Q}_i = \langle c_{\mathcal{Q}_i}, G_{\mathcal{Q}_i} \rangle_Z \subset \mathbb{R}$ and a target $y_i \in \mathbb{R}$, the set-based regression loss is defined as*

$$E_{Reg}(y_i, \mathcal{Q}_i) = \underbrace{\frac{1}{2}(c_{\mathcal{Q}_i} - y_i)^2}_{\text{standard training loss}} + \underbrace{\frac{\eta_Q}{\epsilon}\ \mathrm{lnDia}(G_{\mathcal{Q}_i})}_{\text{volume loss}},$$

with weighting factor $\eta_Q \in \mathbb{R}$ and perturbation radius $\epsilon \in \mathbb{R}_+$. The gradient of $E_{Reg}(y_i, \mathcal{Q}_i)$ w.r.t. $\mathcal{Q}_i$ is computed by:

$$\nabla_{\mathcal{Q}_i} E_{Reg}(y_i, \mathcal{Q}_i) = \Big\langle c - y_i, \frac{\eta_Q}{\epsilon}\ \mathrm{lnDia}'(G_{\mathcal{Q}_i}) \Big\rangle_Z .$$

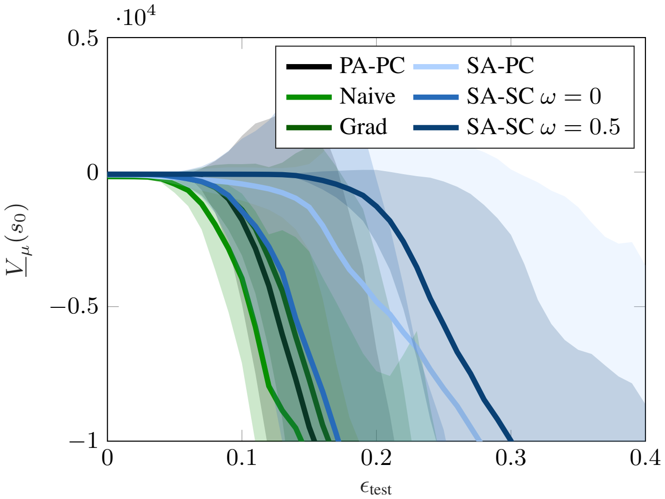

Figure 7: Comparison of $\underline{V}_\mu(s_0)$ for Quadrocopter 2D.

Proof. The gradient w.r.t. a zonotope is represented as a zonotope as well, consisting of the gradient w.r.t. the center and the gradient w.r.t. the generator matrix (Koller, Ladner, and Althoff 2024, Def. 8). Hence, the gradient of the set-based regression is:

$$\begin{aligned} &\nabla_{\mathcal{Q}_i} E_{Reg}(y_i, \mathcal{Q}_i) \\ &\quad = \big\langle \nabla_{c_{\mathcal{Q}_i}} E_{Reg}(y_i, \mathcal{Q}_i), \nabla_{G_{\mathcal{Q}_i}} E_{Reg}(y_i, \mathcal{Q}_i) \big\rangle_Z \\ &\quad = \Big\langle c_{\mathcal{Q}_i} - y_i, \frac{\eta_Q}{\epsilon}\ \mathrm{diag}(|G_{\mathcal{Q}_i}|\,\mathbf{1})^{-1}\ \mathrm{sgn}(G_{\mathcal{Q}_i}) \Big\rangle_Z . \\ &\quad = \Big\langle c_{\mathcal{Q}_i} - y_i, \frac{\eta_Q}{\epsilon}\ \mathrm{lnDia}'(G_{\mathcal{Q}_i}) \Big\rangle_Z . \qquad \square \end{aligned}$$

C Proofs of Sec. 4

Proposition 3. *Given a neural network and an input set $\mathcal{H}_{k-1}$ with the enclosing interval $[l_{k-1}, u_{k-1}] \supseteq \mathcal{H}_{k-1}$, the expected value of the enclosure of the output set of the k-th layer is*

$$\mathbb{E}_{h_k \sim \mathcal{H}_k}[h_k] = \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[L_k(h_{k-1})],$$

with $\mathcal{H}_k = \texttt{enclose}(L_k, \mathcal{H}_{k-1})$. Moreover, for neural networks with only ReLU*-activations, the approximation errors are minimal with approximation slope $m_k = (\sigma_k(u_{k-1}) - \sigma_k(l_{k-1}))/(u_{k-1} - l_{k-1})$.*

Proof. We split cases on the type of layer L_k.

Case (1): Linear layer The expected value is preserved by linearity of the expectation:

$$\begin{aligned} \mathbb{E}_{h_k \sim \mathcal{H}_k}[h_k] &\overset{(17)}{=} c_k = W_k\, c_{k-1} + b_k \\ &= W_k\, \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[h_{k-1}] + b_k \\ &= \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[W_k\, h_{k-1} + b_k] \\ &= \mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[L_k(h_{k-1})]. \end{aligned}$$

Case (2): ReLU layer Activation functions are applied element-wise, thus we consider each dimension individually; to avoid clutter, we drop the dimension index $x_{(i)}$.

We distinguish between three cases: (2a) $l_{k-1}, u_{k-1} \leq 0$, (2b) $l_{k-1}, u_{k-1} \geq 0$, (2c) $l_{k-1} < 0 < u_{k-1}$. For cases (i) and (ii), ReLU is linear, thus by linearity of the expectation the expected value is preserved. For case (iii), we approximate the ReLU activation function with the affine map $m_k\, x + t_k$. We first derive a condition to ensure preserving the expected value. After that we show the slope that minimizes the approximation errors is:

$$m_k = \frac{\mathrm{ReLU}(u_{k-1}) - \mathrm{ReLU}(l_{k-1})}{u_{k-1} - l_{k-1}} = \frac{u_{k-1}}{u_{k-1} - l_{k-1}}. \tag{35}$$

The expected value is preserved if the offset t_k to satisfies the condition:

$$\begin{aligned} & & \mathbb{E}_{h_k \sim \mathcal{H}_k}[h_k] &\overset{!}{=} \mathbb{E}[\mathrm{ReLU}(h_{k-1})] \\ &\iff & c_k &= \mathbb{E}[\mathrm{ReLU}(h_{k-1})] \\ &\iff & m_k\, c_{k-1} + t_k &= \mathbb{E}[\mathrm{ReLU}(h_{k-1})] \\ &\iff & t_k &= \mathbb{E}[\mathrm{ReLU}(h_{k-1})] \\ & & &\quad - m_k\, c_{k-1}, \end{aligned} \tag{36}$$

with $h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})$. Hence, we fix the offset t_k w.r.t. the slope m_k. Moreover, to find the optimal slope m_k, we compute the expected value $\mathbb{E}[\mathrm{ReLU}(h_{k-1})]$ using the probability density function $f_{\mathcal{H}_k}$ for the distribution of $\mathrm{ReLU}(\mathscr{U}(l_{k-1}, u_{k-1}))$. Therefore, we first compute the probability mass below 0 using the cumulative distribution function (Hinton and Ghahramani 1997; Socci, Lee, and Seung 1997):

$$\begin{aligned} F_{\mathscr{U}(l_{k-1}, u_{k-1})}(0) &= \int_{-\infty}^{0} f_{\mathscr{U}(l_{k-1}, u_{k-1})}(h_{k-1})\, \mathrm{d}h_{k-1} \\ &= \int_{l_{k-1}}^{0} \frac{1}{u_{k-1} - l_{k-1}}\, \mathrm{d}h_{k-1} \\ &= \frac{-l_{k-1}}{u_{k-1} - l_{k-1}}. \end{aligned}$$

The probability mass is concentrated as a peak at zero using the Dirac delta $\delta(x)$ (Au and Tam 1999):

$$\delta(x) = \begin{cases} \infty & x = 0, \\ 0 & \text{else}, \end{cases} \qquad \int_{-\infty}^{\infty} f(x)\, \delta(x)\, \mathrm{d}x = f(0).$$

Thus, the probability density function is:

$$f_{\mathcal{H}_k}(h_k) = \begin{cases} \frac{1 - l_{k-1}\, \delta(h_k)}{u_{k-1} - l_{k-1}}, & 0 \leq h_k < u_{k-1}, \\ 0 & \text{otherwise} \end{cases}$$

Thus, the expected value of the transformed distribution is:

$$\begin{aligned} &\mathbb{E}_{h_{k-1} \sim \mathscr{U}(l_{k-1}, u_{k-1})}[\mathrm{ReLU}(h_{k-1})] \\ &= \int_{-\infty}^{\infty} \mathrm{ReLU}(h_{k-1})\, f_{\mathcal{H}_k}(\mathrm{ReLU}(h_{k-1}))\, \mathrm{d}h_{k-1} \\ &= \int_{0}^{u_{k-1}} h_k\, f_{\mathcal{H}_k}(h_k)\, \mathrm{d}h_k \\ &= \int_{0}^{u_{k-1}} h_k\, \frac{1 - l_{k-1}\, \delta(h_k)}{u_{k-1} - l_{k-1}}\, \mathrm{d}h_k \\ &= \left. \frac{h_k^2}{2\,(u_{k-1} - l_{k-1})} \right|_{0}^{u_{k-1}} = \frac{u_{k-1}^2}{2\,(u_{k-1} - l_{k-1})}. \end{aligned}$$

Hence,

$$\begin{aligned} t_k &\overset{(36)}{=} \frac{u_{k-1}^2}{2\,(u_{k-1} - l_{k-1})} - m_k\, c_{k-1} \\ &\overset{(7)}{=} \frac{1}{2} \left(\frac{u_{k-1}^2}{(u_{k-1} - l_{k-1})} - m_k\, (u_{k-1} + l_{k-1}) \right). \end{aligned} \tag{37}$$

We now find the slope m_k that minimizes the approximation errors; by satisfying (36) we ensure preserving the expected value. For concise notation, we abbreviate the approximation error at x with slope m_k by

$$d_x(m_k) \coloneqq |(m_k\, x + t_k) - \mathrm{ReLU}(x)|. \tag{38}$$

We optimize the slope m_k for minimal approximation errors:

$$\min_{m_k} \max_{x \in [l_{k-1}, u_{k-1}]} d_x(m_k).$$

With (Koller, Ladner, and Althoff 2024, Prop. 7), we know that the approximation error are located at $x \in \{l_{k-1}, 0, u_{k-1}\}$; hence, we rewrite:

$$\min_{m_k} \max_{x \in \{l_{k-1}, 0, u_{k-1}\}} d_x(m_k).$$

From $l_{k-1} < 0 < u_{k-1}$, we can simplify

$$\begin{aligned} d_{l_{k-1}}(m_k) &= |m_k\, l_{k-1} + t_k|, \\ d_0(m_k) &= |t_k|, \\ d_{u_{k-1}}(m_k) &= |m_k\, u_{k-1} + t_k - u_{k-1}|. \end{aligned} \tag{39}$$

Moreover, we know that that at least one approximation error is greater 0:

$$d_{l_{k-1}}(m_k) > 0 \lor d_0(m_k) > 0 \lor d_{u_{k-1}}(m_k) > 0.$$

Hence, the optimal slope m_k is located at an intersection of two error functions $d_{x_1}(m_k) = d_{x_2}(m_k) \geq d_{x_3}(m_k)$ with $\{x_1, x_2, x_3\} = \{l_{k-1}, 0, u_{k-1}\}$. Thus, for each intersection we find the optimal slope:

Case (2c.i): $d_{l_{k-1}}(m_k) = d_0(m_k)$

$$\begin{aligned} & & d_0(m_k) &= d_{l_{k-1}}(m_k) \\ &\overset{(39)}{\iff} & |t_k| &= |m_k\, l_{k-1} + t_k| \\ &\iff & t_k^2 &= (m_k\, l_{k-1} + t_k)^2 \\ &\iff & 0 &= (m_k\, l_{k-1})^2 + 2\, m_k\, l_{k-1}\, t_k \\ &\overset{(37)}{\iff} & 0 &= m_k^2 - \frac{u_{k-1}}{u_{k-1} - l_{k-1}}\, m_k \\ &\iff & m_k &= 0 \lor m_k = \frac{u_{k-1}}{u_{k-1} - l_{k-1}}. \end{aligned}$$

Case (2c.ii): $d_{l_{k-1}}(m_k) = d_{u_{k-1}}(m_k)$

$$\begin{aligned} & & d_{l_{k-1}}(m_k) &= d_{u_{k-1}}(m_k) \\ &\overset{(39)}{\iff} & |m_k\, l_{k-1} + t_k| &= |m_k\, u_{k-1} + t_k - u_{k-1}| \\ &\iff & (m_k\, l_{k-1} + t_k)^2 &= (m_k\, u_{k-1} + t_k - u_{k-1})^2 \\ &\overset{(37)}{\iff} & m_k\, u_{k-1}\, l_{k-1} &= u_{k-1}\, \frac{u_{k-1}^2}{u_{k-1} - l_{k-1}} - u_{k-1}^2 \\ &\iff & m_k &= \frac{u_{k-1}}{u_{k-1} - l_{k-1}}. \end{aligned}$$

Case (2c.iii): $d_0(m_k) = d_{u_{k-1}}(m_k)$

$$\begin{aligned}
& & d_0(m_k) &= d_{u_{k-1}}(m_k) \\
&\overset{(39)}{\iff} & |t_k| &= |m_k\, u_{k-1} + t_k - u_{k-1}| \\
&\iff & t_k^2 &= (m_k\, u_{k-1} + t_k - u_{k-1})^2 \\
&\overset{(37)}{\iff} & -m_k^2\, l_{k-1} + m_k \left(\frac{l_{k-1}\,(2\, u_{k-1} - l_{k-1})}{u_{k-1} - l_{k-1}} \right) & \\
& & &= \frac{u_{k-1}^2}{u_{k-1} - l_{k-1}} + u_{k-1} \\
&\iff & m_k = 1 \vee m_k &= \frac{u_{k-1}}{u_{k-1} - l_{k-1}}.
\end{aligned}$$

In each case, we show (35).

□