IEEE CSS

Robustness Analysis of Neural Networks via Efficient Partitioning With Applications in Control Systems

Michael Everett, Golnaz Habibi, *Member, IEEE*, and Jonathan P. How, *Fellow, IEEE*

Abstract—**Neural networks (NNs) are now routinely implemented on systems that must operate in uncertain environments, but the tools for formally analyzing how this uncertainty propagates to NN outputs are not yet commonplace. Computing tight bounds on NN output sets (given an input set) provides a measure of confidence associated with the NN decisions and is essential to deploy NNs on safety-critical systems. Recent works approximate the propagation of sets through nonlinear activations or partition the uncertainty set to provide a guaranteed outer bound on the set of possible NN outputs. However, the bound looseness causes excessive conservatism and/or the computation is too slow for online analysis. This letter unifies propagation and partition approaches to provide a family of robustness analysis algorithms that give tighter bounds than existing works for the same amount of computation time (or reduced computational effort for a desired accuracy level). Moreover, we provide new partitioning techniques that are aware of their current bound estimates and desired boundary shape (e.g., lower bounds, weighted ℓ_∞-ball, convex hull), leading to further improvements in the computation-tightness tradeoff. The letter demonstrates the tighter bounds and reduced conservatism of the proposed robustness analysis framework with examples from model-free RL and forward kinematics learning.**

Index Terms—**Neural networks, uncertain systems, robotics.**

I. Introduction

NEURAL networks (NNs) are ubiquitous across robotics for perception, planning, and control tasks. While empirical performance statistics can indicate that a NN has learned a useful input-output mapping, there are still concerns about how much confidence to associate with decisions resulting from a learned system. One direction toward providing a confidence measure is to consider how the various sources of

uncertainty in training/execution processes map to uncertainty in outputs of trained NNs. Many of these uncertainties appear at the NN input (e.g., from noisy/adversarially attacked sensing, unknown initial conditions), thus this letter focuses on the problem of propagating input uncertainties through NNs to bound the set of possible NN outputs online.

Analysis of how a set of possible inputs propagates through a NN has an inherent tradeoff between computation time and conservatism. Exact methods [1]–[6] are computationally intractable for online analysis, so we focus on finding guaranteed outer bounds on the network outputs. Most existing methods propagate the entire input set through the NN – we refer to these as *Propagators* [7]–[12].

Although some of these propagators scale to high dimensional NNs, large input sets (e.g., from high state uncertainty) induce massive conservatism, even for small NNs. *Partitioners* [13]–[17] are a promising direction toward propagating large input sets through NNs, particularly when the number of uncertain NN inputs is relatively small, as in many control systems. Nonetheless, current partitioners spend excessive computational effort when refining cells and suffer from simple propagation strategies.

The contributions of this letter are: (i) a family of analysis tools that provide tighter guaranteed bounds on possible NN outputs for the same computational effort compared to the state-of-the-art, (ii) two new partitioning algorithms that efficiently refine the input set partition based on desired output set shapes, (iii) demonstrations of the proposed algorithms on NNs used in practice, including various NN architectures (e.g., fully connected, LSTM), deep NNs, and NNs with various nonlinear activations (e.g., ReLU, tanh), and (iv) applications on control systems to improve a robust RL agent's navigation efficiency by 22% and reduce reachable set estimation error by 96% for a robot arm.

II. Background and Related Work

Problem Statement: Given a trained NN and a set of possible NN inputs, the objective is to find the tightest guaranteed over-approximation of the set of possible NN outputs. The exact set of possible outputs is called the NN's *reachable set*. In general, finding the exact reachable set, $\mathcal{U}$, is computationally intractable for reasonably sized NNs; instead, the goal is to

compute the over-estimate $\mathcal{U}_e$ such that $\mathcal{U} \subseteq \mathcal{U}_e$ for a given input set.

A. Propagators

Propagators estimate how the full input set moves through the network, and they primarily differ in approximation strategies of the nonlinear activation functions. At one extreme, Interval Bound Propagation (IBP) [7] approximates the output of each layer with a tight ℓ_∞ ball, leading to conservative but fast-to-compute bounds of the final layer. Convex relaxation-based techniques [18] often achieve tighter bounds with more computation by approximating nonlinear activations with linear bounds – some of these can be solved in closed-form [10], [11]. Other propagators provide tighter analysis at the cost of higher computation time, including approaches based on QP/SDP [8], [9], and convex relaxation refinements [19]. While this letter focuses on analysis of trained NNs, several recent works consider the orthogonal problem of how to use these propagation techniques during training [20].

B. Partitioners

Partitioners break the input set into smaller regions, compute the reachable set of each small region, and return the total reachable set as the union of each smaller region's reachable set. The idea is depicted in Fig. 2 for a simple NN with linear activations. In the top row (without partitioning) IBP operates on the full input set, leading to excessive conservatism in the final output bound (top right: large red dashes vs. red shaded region). The bottom row shows how IBP on two halves of the input set leads to a tighter approximation of the output set.

The key difference between partitioning approaches is the strategy for how to split the input set. Some works make one bisection of the input set [13]; [14] splits the input set into a uniform grid; [15] uses gradients to decide which cells to split for ReLU NNs. [16] improves on [15] using "shadow prices" to optimize *how* to split a particular cell (i.e., along which dimension), but does not provide a way of choosing *which* cells to split when computing tight reachable sets. As illustrated in [17], substantial performance improvements can be achieved by stopping the refinement of cells that are already sufficiently refined. Thus, the current state-of-art partitioner, a Simulation-Guided approach (SG) [17], uses a partitioning strategy where Monte Carlo samples of the exact NN output are used as guidance for efficient partitioning of the input set, reducing the amount of computation required for the same level of bound tightness. SG used IBP to compute output sets, and the two ideas of Partitioners and Propagators have been developed separately toward a similar objective. This letter addresses key gaps in the partitioning literature: we unify Partitioners with state-of-art Propagators for better performance, propose new partitioners that are flexible in the desired output set shape. We then show how improvements in robustness analysis map directly to reduced conservatism in control tasks.

III. APPROACH

This section introduces the overall architecture, describes our new partitioning algorithms, then analyzes the reduction
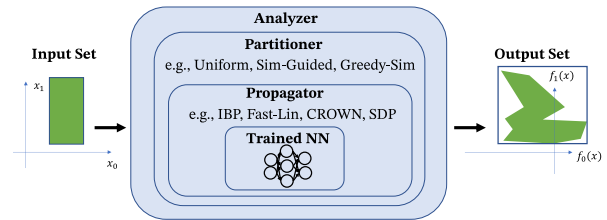

Fig. 1. Robustness Analysis Architecture. This letter efficiently bounds the set of NN outputs for a given input set.
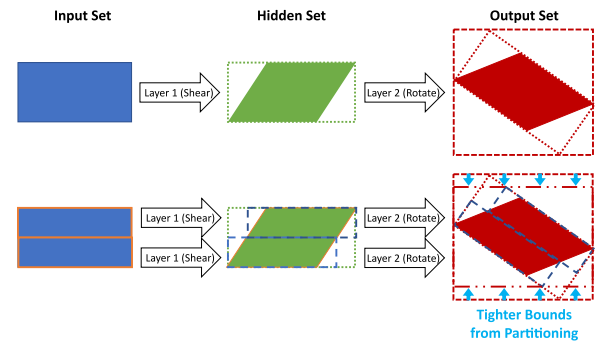

Fig. 2. Partitioner Intuition. (Top) Large input sets cause loose bounds on NN output sets, even for this simple 2-layer NN with linear activations. (Bottom) Tighter bounds can be achieved by partitioning the input set, propagating each cell through the NN, and merging the output sets [14], [17].

in conservatism from partitioning. Figure 1 shows a schematic of the proposed framework with its three nested modules: Analyzer, Partitioner and Propagator. The Analyzer is aware of the desired output shape (e.g., lower bounds, ℓ_∞-ball, convex hull) and termination condition (e.g., computation time, number of Propagator calls, improvement per step). The Analyzer specifies a Propagator (e.g., CROWN [10], IBP [7], SDP [9], Fast-Lin [11]) and a Partitioner (e.g., Uniform [14], Sim-Guided [17] or the algorithms proposed in this section). The Partitioner decides how to split the input set into cells, and the Propagator is used by the Partitioner to estimate the output set corresponding to an input set cell.

A. Greedy Simulation-Guided Partitioning

The state-of-art partitioning algorithm, SG [17], tightens IBP's approximated boundary with the following key steps: (1) acquire N Monte Carlo samples of the NN outputs to under-approximate the reachable set as the interval $[u_{\text{sim}}]$ (where $[\,\cdot\,]$ denotes a closed n-dimensional interval), (2) using IBP, compute the reachable set of the full input set and add this set to a stack M, and (3) (iteratively) pop an element from M, and either stop refining that cell if its computed reachable set is within $[u_{\text{sim}}]$, or bisect the cell, compute each bisection's reachable set, and add both to the stack. The SG algorithm terminates when one of the cell's dimensions reaches some threshold, and the returned reachable set estimate is the weighted ℓ_∞-ball that surrounds the union of all of the cells remaining on the stack and $[u_{\text{sim}}]$.

We propose a partitioning algorithm with better bound tightness for the same amount of computation, called Greedy-Sim-Guided (GSG), described in Algorithm 1, by modifying the choice of which cell in M to refine at each step. Rather than popping the first element from the stack (LIFO) as in SG,

---

Algorithm 1: Greedy Simulation-Guided Partitioning

1   **Input:** propagator operator ($[.]$), termination condition ($\mathcal{T}_c()$), input interval ($[\eta] \subseteq \mathcal{H}$), Adaptive flag ($a = 1$, if AGSG is active), Neural Network Φ

2   **Output:** output boundary $\mathcal{U}_e$
    // Initialization and MC sampling

3   $\mathbf{u}_{sim,n} = \Phi(x_{sim,n} \in \mathcal{H}), n = 1, \cdots, N$

4   $[u_{sim}] \leftarrow$ extrema of $u_{sim,n}$ // sim boundary

5   $M \leftarrow \emptyset, \mathcal{U}_e \leftarrow \emptyset$

6   **if** a **then**
     // AGSG: Adaptive Initialization

7      $(\eta^*, u^*) \leftarrow \arg\min_{(\eta,u)} \|u - \mu([u_{sim}])\|_2$

8      $\eta_e = Expand(\eta^*)$

9      $\{[\eta']_i\} = Decompose(\{[\eta] \setminus [\eta_e]\})$ //   Fig. 3b

10   **for** $[\eta']_i \in \{[\eta']_i\}$ **do**

11        $[u_i] = [\Phi]([\eta]_i)$

12        $M \leftarrow M \cup \{([\eta]_i, [u_i])\}\}$

13   **else**
     // GSG: M is full interval

14      $[u] = [\Phi]([\eta])$

15      $M \leftarrow M \cup \{([\eta], [u])\}$
    // Partition Refinement

16   **while** $M \neq \emptyset$ **do**
     // Fig. 3a (Pop from M)

17      $([\eta_w], [u_w]) \leftarrow \arg\max_{([\eta],[u]) \in M} d([u], [u_{\text{sim}}])$

18   **if** $[u_w] \in [\mathbf{u}_{sim}]$ **then**

19        $\mathcal{U}_e \leftarrow \mathcal{U}_e \cup [u_w]$

20   **else**

21   **if** $\mathcal{T}_c()$ **then**

22   break // Terminate

23   **else**

24          $[\eta_1], [\eta_2] = \text{Bisect}([\eta_w])$

25          $[u_1], [u_2] = [\Phi]([\eta_{i=1,2}])$

26          $M \leftarrow M \cup \{([\eta_1], [u_1])\} \cup \{([\eta_2], [u_2])\}$

27   **return** $\mathcal{U}_e \leftarrow \mathcal{U}_e \cup (\cup_{\{[\eta],[u]\} \in M^{[u]}})$

GSG refines the input cell with corresponding output range that is furthest outside the output boundary of the N samples (Line 17). This is illustrated in Fig. 3a, where the input cell corresponding to d_2 would be refined before d_1, because d_2's output set (magenta) further exceeds the simulation-guided boundary estimate (black rectangle surrounding the black NN samples).

Whereas SG might choose a cell that is not pushing the overall boundary outward at a given iteration, GSG will always choose to refine an input cell that is pushing the boundary. This heuristic gives the opportunity to reduce the boundary estimate at each iteration. While the core SG algorithm remains the same, the greedy strategy can greatly improve the algorithm's performance.

B. Adaptive-Greedy-Simulation-Guided Partitioning

The Adaptive-Greedy-Sim-Guided (AGSG) algorithm extends GSG's initialization procedure to reduce wasted

---

Algorithm 2: $Expand([\eta]^{(0)})$

1   **Input:** Initial input interval($[\eta]^{(0)}$), step size(e_s), NN propagator $[\Phi]$, simulated output set$[\mathbf{u}_{sim}]$

2   **Output:** expanded interval $[\eta_e]$

3   $[\eta_e] \leftarrow [\eta]^{(0)}$

4   $[u] \leftarrow [\Phi]([\eta_e])$

5   **while** *True* **do**

6   **if** $[u] \subset [u_{sim}]$ **then**

7        $[\eta_e] \leftarrow [\eta_e] + e_s$

8        $[u] = [\Phi]([\eta_e])$

9   **else**

10       $[\eta_e] \leftarrow [\eta_e] - e_s$

11   **return** $[\eta_e]$

(a) GSG Set Selection   (b) AGSG Decomposition

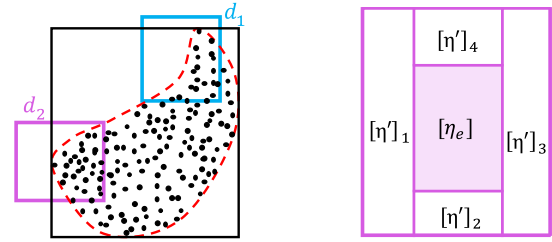Fig. 3. (a) GSG selects from candidates in M: the input set corresponding to the magenta output set is chosen for refinement, because $d_2 > d_1$ (where $[u_{\text{sim}}]$ is the black box). (b) AGSG decomposition $[\eta] \setminus [\eta_e]$ into four new intervals $[\eta']_{1,2,3,4}$.

---

computation time refining unimportant input regions. When a is activated, the AGSG initialization process is used (Lines 6-12). After computing $[u_{sim}]$, AGSG initializes $[\eta_e]$ as the input point whose output is at the middle of simulated outputs. This cell is repeatedly expanded with step size e_s as long as its output set (computed by a Propagator) remains inside $[u_{sim}]$. The Expand procedure is explained in Algorithm 2. The expanding cell is guaranteed to produce an output inside the simulated boundary.

Lemma 1: The overestimated output of the expanded interval $[\eta_e] \in \mathcal{H}$ to the neural network Φ when the output set is approximated by a propagator $[\Phi]$, is $[u_e]$ such that

$$[u_e] = [\Phi]([\eta_e]) \subseteq [u_{sim}]. \tag{1}$$

Proof: Assume the expanded interval and its output estimate at step t are denoted by $[\eta_e]^t$ and $[u_e]^t$ respectively. If $[u_e]^t = [\Phi]([\eta_e]^t) \not\subset [u_{sim}]$, there are two possible cases: (1) $[u_e]^{t-1} \subset [u_{sim}]$ (the expansion of $[\eta_e]^{t-1}$ to $[\eta_e]^t$ causes this outcome). In this case, $[\eta_e]^t$ is reduced back to $[\eta_e]^{t-1}$ according to Algorithm 2's expanding condition in line (9). Thus, $[\Phi]([\eta_e]^{t-1}) \subseteq [u_{sim}]$ – this condition is not stable. Case 2 would occur when $[u_e]^{t-1} \not\subset [u_{sim}]$, but this is not possible, since the interval would never be expanded if $[u_e]^{t-1} \not\subset [u_{sim}]$, unless the initial interval's output is outside the simulated boundary: $[u_e]^{(0)} = [\Phi]([\eta_e]^{(0)})$. The initial interval's output cannot be outside the simulated boundary (contradiction), since $[\eta_e]^{(0)}$ is initialized via sampled inputs, thus $u^{(0)} \subset [u_{sim}]$. Therefore the approximated output of the expanding input is always inside the simulated set.   ∎
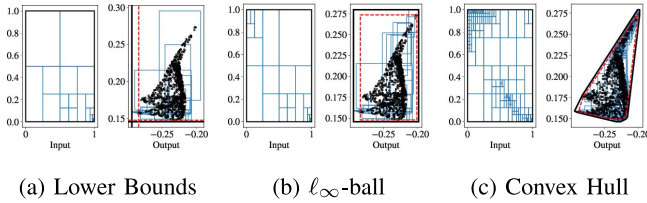
(a) Lower Bounds   (b) ℓ_∞-ball   (c) Convex Hull

Fig. 4. Input & output sets for different output set shapes. The estimated bounds (black) are "tight" when they are close to the bounds from exhaustive sampling (dashed red). The GSG partitioner with CROWN [10] propagator ran for 2 sec.

The remaining input is decomposed into a set of disjoint intervals $[\eta']_i$. Fig. 3 shows this decomposition in 2D, which creates four new intervals $[\eta']_{1,2,3,4}$ (some of which could be empty) of rectangle shape.[1] The new intervals $[\eta']_i$ are passed to GSG as the initial M. The output set boundary estimate returned by AGSG merges that GSG output and the initial expanded cell's output boundary.

C. Boundary Specification

While SG only computes a ℓ_∞-ball over-approximation, GSG/AGSG optimize for the desired output set shape. For example, if the goal is to find a convex hull over-approximation, GSG/AGSG modify the idea from Fig. 3 to select the input set that is furthest from the convex hull boundary (instead of the ℓ_∞-ball, as in SG [17]).

IV. EXPERIMENTAL RESULTS

This section shows example partitions, applies the ideas to a robotic arm task, demonstrates better closed-loop behavior in collision avoidance, shows an ability to scale to various network sizes/architectures, and measures improvement along the time vs. tightness tradeoff.

A. Partitions for Different Output Shapes

The ability to partition efficiently for different output shapes is shown in Fig. 4 for a randomly initialized NN with 2 inputs, 2 outputs, and 50 nodes in hidden layer, i.e., (2, 50, 2), with ReLU activations, and input set $[0, 1] \times [0, 1]$. Each of (a-c) uses GSG with CROWN for 2 seconds. Recall that SG [14] would only return one output set for (a-c).

B. Comparisons to Baselines & Ablation Study

Four partitioning algorithms are compared in Fig. 5 for the same (2, 50, 2) NN and input set. Each analyzer runs for 2 seconds to compute an estimated output set. The true output set is obtained by exhaustively sampling from the input space, and error is reported as percent extra area, $\frac{A_{\text{estimate}} - A_{\text{true}}}{A_{\text{true}}}$. The proposed partitioning algorithms GSG (c) and AGSG (d) use only 280 and 275 partitions respectively, and their approximation error is 0.018, which indicates more than 79% improvement over SG -CROWN (b) and 95% over the state of the art SG-IBP [17]. In addition to quantitative improvement, Fig. 5 illustrates the input set partitions of each algorithm, which highlights how GSG refines different/fewer cells as SG, and that AGSG does not strictly make bisections.

[1]Extension of the decomposition to higher dims. is left as future work.

(a) SG+IBP [17]
Partitions: 1021; Error: 0.35

(b) SG+CROWN
Partitions: 387; Error: 0.09

(c) GSG+CROWN
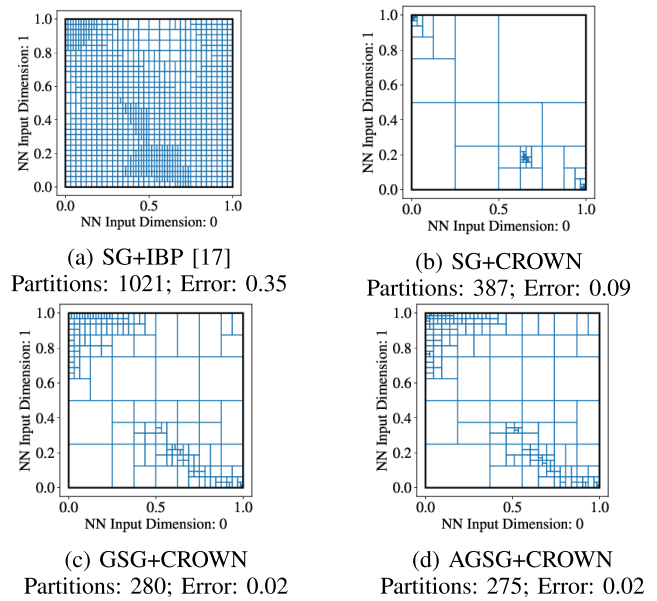Partitions: 280; Error: 0.02

(d) AGSG+CROWN
Partitions: 275; Error: 0.02

Fig. 5. Input partitions for a random (2, 50, 2) ReLU NN analyzed for 2 seconds. New methods (b, c, d) reduce the number of input partitions and output set (convex hull) error.

(a) Robotic arm [14]

Algorithm		Stats	
(Prop. + Part.)	Error	Prop. Calls	Partitions
IBP + SG [17]	0.216	1969	985
IBP + GSG	0.042	869	435
IBP + AGSG	0.040	847	425
Fast-Lin + SG	0.134	593	297
Fast-Lin + GSG	0.009	473	237
Fast-Lin + AGSG	**0.008**	461	232
CROWN + SG	0.134	587	294
CROWN + GSG	0.009	467	234
CROWN + AGSG	**0.008**	**453**	**228**

(b) Comparison of Algorithms

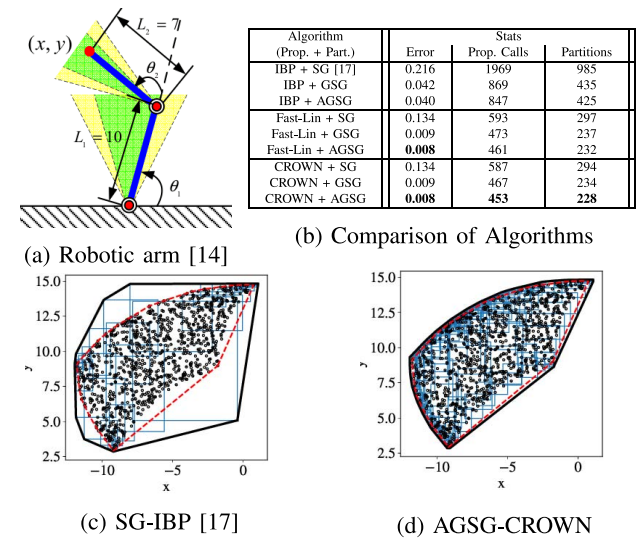(c) SG-IBP [17]

(d) AGSG-CROWN

Fig. 6. Reachable set estimate of a robotic arm. GSG/AGSG-CROWN achieves 96% lower error than [17] (2 sec.).

C. Applications in Robotics and Control

1) Reachable Set Analysis of Robotic Arm: Borrowing the robotic arm model from [14], [17], we compare our algorithm to [17] for reachable set estimation of a forward kinematics model. Fig. 6 shows the 2 DOF robot arm, with input (θ_1, θ_2) as joint angles and output (x, y) as end effector position. The nonlinear dynamics are modeled by a small (2, 5, 2) NN with tanh activations. One motivation for computing tight reachable sets here is to ensure that the robot arm will not collide with any obstacles, without being overly conservative. We assume a time limit of 2 sec to approximate the convex hull of end effector positions from the set of joint angles $(\theta_1, \theta_2) \in [\frac{\pi}{3}, \frac{2\pi}{3}] \times [\frac{\pi}{3}, \frac{2\pi}{3}]$. As shown in Fig. 6a, AGSG-CROWN reduces the error from [17] by 96%. Only switching

(a) CROWN (highly conservative)   (b) GSG-CROWN (less conservative)   (c) Worst-Case Q-Values (Model-Free RL)
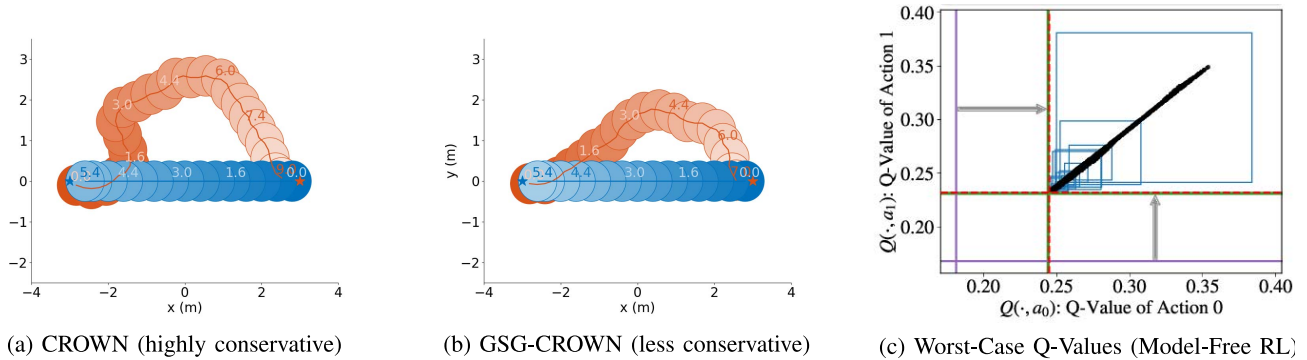
Fig. 7. Multiagent collision avoidance under uncertainty. In (a), a robust but conservative trajectory from a robust RL formulation [21] that used CROWN/Fast-Lin [10] to estimate worst-case Q-values under uncertainty on obstacle positions. In (b), the proposed GSG-CROWN algorithm enables the orange agent to reach its goal faster (7 vs. 9 sec) while still avoiding the blue agent. This improved behavior is a result of tighter estimates of worst-case Q-values, shown at a single timestep in (c) (purple → green).

the partitioner (SG-IBP vs. (A)GSG-IBP) still achieves 80% error reduction. The estimated boundary is shown in (c, d).

2) Multiagent Collision Avoidance: Deep RL methods are popular in multiagent collision avoidance literature [22], [23], but they rarely account for measurement uncertainty. Reference [21] proposed a certifiably robust deep RL algorithm, which involves estimating a tight lower bound on the NN (e.g., DQN) output given that the agent could be within some state set. In that work, large input uncertainties can degrade performance, partially due to overly conservative lower bounds from Fast-Weng *et al.* [11]. This example motivates the need for tight reachable set estimation algorithms, as proposed in this letter.

The robust-but-conservative behavior caused by CROWN/Fast-Lin is shown in Fig. 7a (±0.5m uncertainty on the blue agent's position at each timestep). By instead using GSG-CROWN to estimate worst-case Q-values (Fig. 7b), the orange agent reaches the goal much faster while still avoiding the blue agent. This improved behavior is a result of tighter estimates of worst-case Q-values, shown at a single timestep in Fig. 7c. For this experiment, a (11, 64, 64, 11) DQN (11 states & 11 discrete actions) was trained with perfect measurements in the `gym-collision-avoidance` environment [24]. Furthermore, this application is a case where only *lower bounds* on the NN outputs are needed, which motivates the use of our proposed algorithms that can focus computation toward this objective.

D. Scalability to Larger NNs

The proposed algorithms scale to bigger NNs as shown in Table I and improve on existing methods by orders of magnitude. For a given number of propagator calls (100), the average error (close to 0 is best) of 10 random NNs is reported for 4 different architectures (small, deep, higher dimension, LSTM), and for different boundary types. The (4, 100, 10) NN has uncertainty on the full 4D input set and the LSTM uses 2D uncertainty on the last timestep. While this letter's approaches scale well to deep NNs and various architectures, future work should consider the challenges from settings with even higher dimensional input uncertainties.

E. Computation-Performance Tradeoff

Throughout this letter, we have leveraged the idea of partitioning the input set to tighten the approximated boundary.

TABLE I
APPROXIMATION ERROR (CLOSER TO 0 IS BETTER) FOR FOUR DIFFERENT TYPES OF NNS: SMALL, DEEP, HIGHER DIMENSION, AND LSTM. REPORTED VALUES ARE AVERAGE ERROR ACROSS 10 RANDOMLY INITIALIZED NNS AFTER UP TO 100 PROPAGATOR CALLS

NN	Algorithm (Prop. + Part.)	Boundary Type		
		Lower Bounds	ℓ_∞-ball	Convex Hull
(2, 100, 2)	IBP [7]	1.50	7.77×10^1	9.06
	IBP + SG [17]	2.47×10^{-1}	4.06	1.49
	IBP + GSG	1.70×10^{-1}	3.44	1.44
	Fast-Lin [11]	2.78×10^{-1}	4.62	1.90
	Fast-Lin + SG	1.70×10^{-3}	1.70×10^{-2}	1.12×10^{-1}
	Fast-Lin + GSG	3.94×10^{-3}	5.48×10^{-2}	7.23×10^{-2}
	CROWN [10]	2.15×10^{-1}	3.29	1.55
	CROWN + SG	$\mathbf{1.34 \times 10^{-3}}$	$\mathbf{1.23 \times 10^{-2}}$	1.09×10^{-1}
	CROWN + GSG	3.49×10^{-3}	5.32×10^{-2}	$\mathbf{6.65 \times 10^{-2}}$
	SDP [9]	1.20×10^{-1}	1.90	1.06
(2, 100, 100, 100, 100, 100, 2)	IBP [7]	1.69×10^2	8.17×10^9	1.07×10^5
	IBP + SG [17]	3.32×10^1	3.16×10^8	2.10×10^4
	IBP + GSG	3.07×10^1	2.67×10^8	1.93×10^4
	Fast-Lin [11]	2.32	1.57×10^6	1.48×10^3
	Fast-Lin + SG	2.65×10^{-4}	3.83×10^{-1}	2.91×10^{-1}
	Fast-Lin + GSG	9.32×10^{-5}	2.12×10^{-1}	2.30×10^{-1}
	CROWN [10]	8.96×10^{-1}	2.42×10^5	5.74×10^2
	CROWN + SG	1.61×10^{-4}	2.03×10^{-1}	1.91×10^{-1}
	CROWN + GSG	$\mathbf{5.40 \times 10^{-5}}$	$\mathbf{1.18 \times 10^{-1}}$	$\mathbf{1.65 \times 10^{-1}}$
(4, 100, 10)	IBP [7]	3.11×10^1	1.41×10^{17}	-
	IBP + SG [17]	1.40×10^1	6.89×10^{13}	-
	IBP + GSG	1.33×10^1	4.37×10^{13}	-
	Fast-Lin [11]	6.18	5.95×10^{10}	-
	Fast-Lin + SG	8.24×10^{-1}	7.81×10^3	-
	Fast-Lin + GSG	7.44×10^{-1}	4.59×10^3	-
	CROWN [10]	4.51	4.01×10^9	-
	CROWN + SG	5.60×10^{-1}	8.17×10^2	-
	CROWN + GSG	$\mathbf{5.00 \times 10^{-1}}$	$\mathbf{5.52 \times 10^2}$	-
LSTM ((8, 8), 64, 2)	IBP [7]	1.56×10^{-2}	1.16×10^2	1.13×10^1
	IBP + SG [17]	2.44×10^{-3}	5.55	1.83
	IBP + GSG	$\mathbf{1.80 \times 10^{-3}}$	$\mathbf{4.90}$	$\mathbf{1.80}$

To empirically show that bounds tighten with additional computational effort, we plot several combinations of partitioners and propagators in Fig. 8 over time. Each color corresponds to a propagator (IBP, CROWN, SDP) and each marker corresponds to a partitioner (SG, GSG, AGSG). This result uses the robotic arm model and convex hull boundaries from before, but with ReLU activations.

A first key takeaway is that additional computation time leads to reduced error (increased tightness) Another key takeaway is that our framework provides many algorithms that exceed the performance of previous state-of-art algorithms [9], [10], [17]. Except the blue dashed line [17] and leftmost green/orange points [9], [10], all of the options are new algorithms proposed by this letter. The analysis provided in the plots informs the choice of propagator and partitioner for

(a) Propagator: IBP

(b) Propagator: CROWN

(c) Propagator: SDP

(d) All Propagators (overlayed)
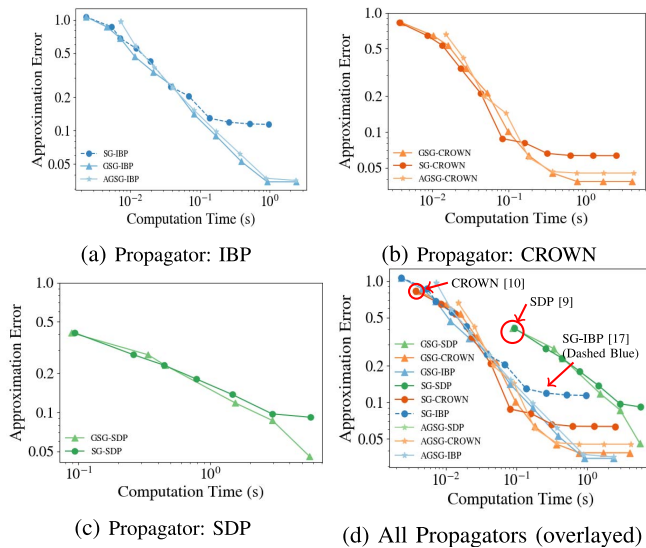
Fig. 8. Improvement in Computation Time vs. Accuracy Tradeoff. (a-c) each compare different partitioners for a single propagator, (d) overlays (a-c) in one figure. Colors indicate Propagator; markers indicate Partitioner. The GSG/AGSG approaches outperform the SG approaches as the bounds are refined. This letter unified partition and propagation ideas to give many new methods (everything without a red annotation) that exceed the state-of-the-art.

a particular application with, say, a desired level of accuracy or budgeted resources (memory/computation). Overall, for this task GSG-CROWN almost always provides the best accuracy vs. computation time tradeoff, requiring $\sim 5\times$ less computation for the same accuracy as SG-IBP [17].

V. CONCLUSION

This letter proposed a suite of algorithms for online robustness analysis of NNs that can provide confidence in NN decisions under uncertainty. We build on recent work for handling large uncertainties by proposing new, flexible partitioning algorithms and give theoretical rationale for partitioning as a strategy for reducing conservatism. Furthermore, we show how recent methods that efficiently relax NN nonlinearities can be unified with partitioning in a single framework, which provides many new state-of-art algorithmic choices for robotics applications. Along with showing improved aggregate performance on random NNs with various sizes/architectures, we show how these ideas can be applied to other learning tasks for control systems, showing a 22% improvement in robust RL for multiagent collision avoidance and a 96% reduction in conservatism for a learned robotic arm kinematic model.

REFERENCES

[1] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Proc. Int. Symp. Automated Technol. Verification Anal.*, 2017, pp. 269–286.

[2] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Proc. Comput. Aided Verification 29th Int. Conf. CAV*, Heidelberg, Germany, 2017, pp. 97–117.

[3] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *Computer Aided Verification*, R. Majumdar and V. Kunčak, Eds. Cham, Switzerland: Springer, 2017, pp. 3–29.

[4] A. Lomuscio and L. Maganti, "An approach to reachability analysis for feed-forward relu neural networks," 2017. [Online]. Available: arXiv:1706.07351.

[5] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019.

[6] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "AI2: Safety and robustness certification of neural networks with abstract interpretation," in *Proc. IEEE Symp. Security Privacy (SP)*, May 2018, pp. 3–18.

[7] S. Gowal *et al.*, "On the effectiveness of interval bound propagation for training verifiably robust models," 2018. [Online]. Available: arXiv:1810.12715.

[8] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018.

[9] M. Fazlyab, M. Morari, and G. J. Pappas, "Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming," 2019. [Online]. Available: arXiv:1903.01287.

[10] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, and L. Daniel, "Efficient neural network robustness certification with general activation functions," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4939–4948.

[11] T. Weng *et al.*, "Towards fast computation of certified robustness for ReLU networks," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 5273–5282.

[12] G. Singh, T. Gehr, M. Mirman, M. Püschel, and M. Vechev, "Fast and effective robustness certification," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 10802–10813.

[13] B. G. Anderson, Z. Ma, J. Li, and S. Sojoudi, "Tightened convex relaxations for neural network robustness certification," 2020. [Online]. Available: arXiv:2004.00570.

[14] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5777–5783, Nov. 2018.

[15] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *Proc. 27th USENIX Security Symp. (USENIX Security)*, 2018, pp. 1599–1614.

[16] V. Rubies-Royo, R. Calandra, D. M. Stipanovic, and C. Tomlin, "Fast neural network verification via shadow prices," 2019. [Online]. Available: arXiv:1902.07247.

[17] W. Xiang, H.-D. Tran, X. Yang, and T. T. Johnson, "Reachable set estimation for neural network control systems: A simulation-guided approach," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, May 14, 2020, doi: 10.1109/TNNLS.2020.2991090.

[18] H. Salman, G. Yang, H. Zhang, C.-J. Hsieh, and P. Zhang, "A convex relaxation barrier to tight robustness verification of neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 9832–9842.

[19] G. Singh, R. Ganvir, M. Püschel, and M. T. Vechev, "Beyond the single neuron convex barrier for neural network certification," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 15072–15083.

[20] H. Zhang, H. Chen, C. Xiao, S. Gowal, R. Stanforth, B. Li, D. Boning, and C.-J. Hsieh, "Towards stable and efficient training of verifiably robust neural networks," 2019. [Online]. Available: arXiv:1906.06316,

[21] M. Everett, B. Lutjens, and J. P. How, "Certified adversarial robustness for deep reinforcement learning," 2020. [Online]. Available: arXiv:2004.06496,

[22] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2018, pp. 6252–6259.

[23] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2017, pp. 285–292.

[24] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Madrid, Spain, Sep. 2018, pp. 3052–3059.