Forward Reachability Analysis of Neural Feedback Systems

EECE/CS 7268: Verifiable Machine Learning

Michael Everett

1 System Dynamics

Consider a system with:

$$\mathbf{x}_{t+1} = p(\mathbf{x}_t, \mathbf{u}_t) \quad \text{(open-loop dynamics)} \tag{1}$$

$$\mathbf{u}_t = \pi(\mathbf{x}_t) \quad \text{(NN controller)} \tag{2}$$

By substituting the controller equation into the dynamics, we obtain the closed-loop system:

$$\mathbf{x}_{t+1} = p(\mathbf{x}_t, \pi(\mathbf{x}_t)) = f(\mathbf{x}_t, \pi) \quad \text{(closed-loop dynamics)} \tag{3}$$

Figure 1 illustrates this system structure, showing how the neural network controller π and plant dynamics p interact to form the closed-loop dynamics f.

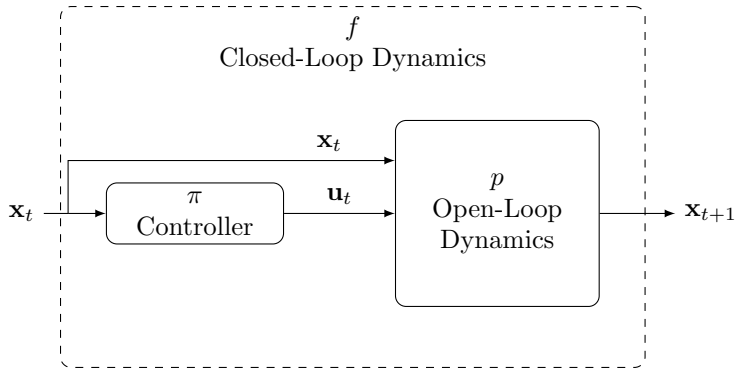

Figure 1: Closed-loop system with neural network controller

Example: $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\pi(\mathbf{x}_t)$

2 Multi-Step Closed-Loop Dynamics

Given the closed-loop dynamics $\mathbf{x}_{t+1} = f(\mathbf{x}_t, \pi)$ and an initial state $\mathbf{x}_t$, we can compute future states by repeatedly applying f:

$$\mathbf{x}_{t+2} = f(\mathbf{x}_{t+1}, \pi) \tag{4}$$

$$= f(f(\mathbf{x}_t, \pi), \pi) \tag{5}$$

$$= f^2(\mathbf{x}_t, \pi) \tag{6}$$

This iterative application of the closed-loop dynamics over multiple timesteps is represented in Figure 2, where each f block corresponds to the dashed box from Figure 1. By chaining these blocks together, we can compute the state evolution over any time horizon T.

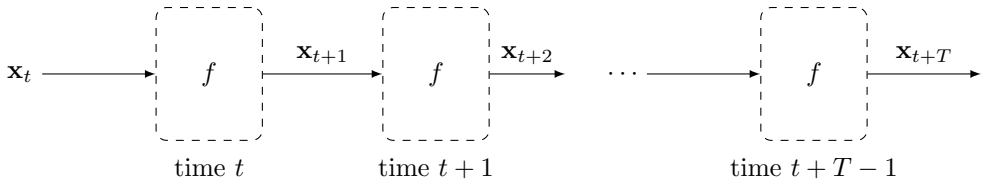

Figure 2: Multi-step closed-loop dynamics over T timesteps, where each f block represents the combined controller and plant dynamics

3 Reachable Sets

When the closed-loop dynamics f is deterministic and the initial state $\mathbf{x}_t$ is known precisely, the future system behavior is perfectly described by a single trajectory. However, in practice, we often only know that the initial state lies within some set $\mathcal{X}_t$ (e.g., due to estimation uncertainty or system variability). In this case, we need to characterize all possible future states the system could reach from any starting condition in $\mathcal{X}_t$. This leads to the concept of reachable sets.

Definition 1. *The reachable set at the initial time t is simply the initial set:*

$$\mathcal{R}_0(\mathcal{X}_t) = \mathcal{X}_t \tag{7}$$

Definition 2. *The 1-step reachable set from a given set $\mathcal{X}_t$ is all states the system could be in at time $t+1$, assuming it starts from some state in $\mathcal{X}_t$:*

$$\mathcal{R}_1(\mathcal{X}_t) = \{f(\mathbf{x}_t, \pi) \mid \forall \mathbf{x}_t \in \mathcal{X}_t\} \tag{8}$$

Definition 3. *The T-step reachable set contains all states reachable after exactly T steps from any initial state in $\mathcal{X}_t$:*

$$\mathcal{R}_T(\mathcal{X}_t) = \{f^T(\mathbf{x}_t, \pi) \mid \forall \mathbf{x}_t \in \mathcal{X}_t\} \tag{9}$$

We sometimes want to know the set of all states reachable within time horizon T, denoted as $\mathcal{R}_{0:T}(\mathcal{X}_t)$, which is the union of reachable sets from time $t = 0$ to $t = T$:

$$\mathcal{R}_{0:T}(\mathcal{X}_t) = \bigcup_{\tau=0}^{T} \mathcal{R}_\tau(\mathcal{X}_t) \tag{10}$$

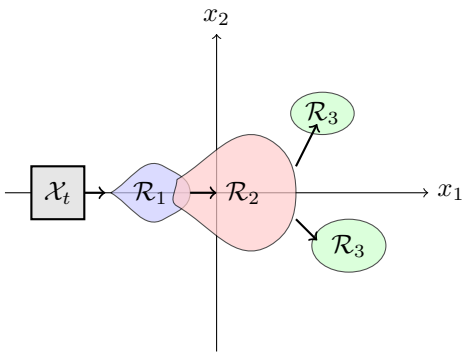

Figure 3: Illustration of reachable sets evolving from left to right. $\mathcal{X}_t$ is the initial set, $\mathcal{R}_1$ and $\mathcal{R}_2$ are connected non-convex sets, while $\mathcal{R}_3$ becomes disconnected.

4 Why Care About $\mathcal{R}_T(\mathcal{X}_t)$?

- If we have a goal region $\mathcal{G}$:

$$\mathcal{R}_T(\mathcal{X}_t) \subseteq \mathcal{G} \Rightarrow \text{system is guaranteed to reach the goal at timestep } T \tag{11}$$

- If we have an avoid region $\mathcal{A}$:

$$\mathcal{R}_{0:T}(\mathcal{X}_t) \cap \mathcal{A} = \emptyset \Rightarrow \text{system is guaranteed to not enter } \mathcal{A} \text{ any time in next } T \text{ steps} \tag{12}$$

5 Computing Reachable Sets

Computing $\mathcal{R}_T$ exactly is intractable (NP-complete) [1].

Instead of computing exact reachable sets, we can calculate over-approximations or bounds, denoted as $\overline{\mathcal{R}}_T$, which are guaranteed to contain the true reachable set: $\mathcal{R}_T \subseteq \overline{\mathcal{R}}_T$. These bounds are valuable because they maintain soundness for verification:

- If $\overline{\mathcal{R}}_T(\mathcal{X}_t) \subseteq \mathcal{G}$, then we can guarantee $\mathcal{R}_T(\mathcal{X}_t) \subseteq \mathcal{G}$, meaning the true system is guaranteed to reach the goal.
- If $\overline{\mathcal{R}}_{0:T}(\mathcal{X}_t) \cap \mathcal{A} = \emptyset$, then we can guarantee $\mathcal{R}_{0:T}(\mathcal{X}_t) \cap \mathcal{A} = \emptyset$, meaning the true system is guaranteed to avoid the obstacle.

To compute such bounds, we formulate optimization problems that find the extremal values of the state in different directions:

$$\max_{\mathbf{x}_t \in \mathcal{X}_t} \mathbf{c}^T \mathbf{x}_{t+T} \quad \text{s.t.} \quad \mathbf{x}_{t+T} = f^T(\mathbf{x}_t, \pi) \tag{13}$$

These optimization problems can be solved using various techniques from neural network verification literature. This connection to neural network verification is natural because the closed-loop system f with a neural controller π forms a computational graph similar to a neural network. The multi-step dynamics f^T can be viewed as unrolling this computational graph T times, analogous to unrolling a recurrent neural network. Consequently, tools developed for neural network verification—such as MILP, LP, CROWN, IBP, and β-CROWN—are directly applicable to bounding reachable sets for neural feedback control systems.

6 Getting Hyper-Rectangle Bounds

To find a hyperrectangle bound on the reachable set $\mathcal{R}_T$, we can choose the objective vectors $\mathbf{c}$ in our optimization problem to maximize or minimize each component of the state vector $\mathbf{x}_{t+T}$. This allows us to compute the tightest hyperrectangle (axis-aligned box) that contains the reachable set.

For example, to get the upper bound on the first component of the state, $\overline{x}_{t+T,1}$, we can let:

$$\mathbf{c}^T = [1\ 0\ \cdots\ 0] \tag{14}$$

In other words, we are solving the following optimization problem:

$$\overline{x}_{t+T,1} = \max_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,1} \quad \text{s.t.} \quad \mathbf{x}_{t+T} = f^T(\mathbf{x}_t, \pi) \tag{15}$$

This same logic can be repeated for each element of the state to get the upper bound vector $\overline{\mathbf{x}}_{t+T}$:

$$\mathbf{c}^T = [1\ 0\ \cdots\ 0] \quad \text{to get } \overline{x}_{t+T,1} = \max_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,1} \tag{16}$$

$$\mathbf{c}^T = [0\ 1\ \cdots\ 0] \quad \text{to get } \overline{x}_{t+T,2} = \max_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,2} \tag{17}$$

$$\vdots \tag{18}$$

$$\mathbf{c}^T = [0\ 0\ \cdots\ 1] \quad \text{to get } \overline{x}_{t+T,n} = \max_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,n} \tag{19}$$

The upper bound vector is then constructed as:

$$\overline{\mathbf{x}}_{t+T} = [\overline{x}_{t+T,1}, \overline{x}_{t+T,2}, \ldots, \overline{x}_{t+T,n}]^T \tag{20}$$

Then, we can flip the sign of the objective vector to get the lower bound vector $\underline{\mathbf{x}}_{t+T}$. For instance, to get the lower bound on the first component, $\underline{x}_{t+T,1}$, we use $\mathbf{c}^T = [-1\ 0\ \cdots\ 0]$ in our maximization framework:

$$\max_{\mathbf{x}_t \in \mathcal{X}_t} \mathbf{c}^T \mathbf{x}_{t+T} = \max_{\mathbf{x}_t \in \mathcal{X}_t} (-1) \cdot x_{t+T,1} = -\min_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,1} \tag{21}$$

Thus, $\underline{x}_{t+T,1} = \min_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,1}$. Repeating for all components:

$$\mathbf{c}^T = [-1\ 0\ \cdots\ 0] \quad \text{to get} \quad -\underline{x}_{t+T,1} = \max_{\mathbf{x}_t \in \mathcal{X}_t} (-x_{t+T,1}) \Rightarrow \underline{x}_{t+T,1} = \min_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,1} \tag{22}$$

$$\mathbf{c}^T = [0\ -1\ \cdots\ 0] \quad \text{to get} \quad -\underline{x}_{t+T,2} = \max_{\mathbf{x}_t \in \mathcal{X}_t} (-x_{t+T,2}) \Rightarrow \underline{x}_{t+T,2} = \min_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,2} \tag{23}$$

$$\vdots \tag{24}$$

$$\mathbf{c}^T = [0\ 0\ \cdots\ -1] \quad \text{to get} \quad -\underline{x}_{t+T,n} = \max_{\mathbf{x}_t \in \mathcal{X}_t} (-x_{t+T,n}) \Rightarrow \underline{x}_{t+T,n} = \min_{\mathbf{x}_t \in \mathcal{X}_t} x_{t+T,n} \tag{25}$$

The lower bound vector is constructed as:

$$\underline{\mathbf{x}}_{t+T} = [\underline{x}_{t+T,1}, \underline{x}_{t+T,2}, \ldots, \underline{x}_{t+T,n}]^T \tag{26}$$

Once we have computed these bounds, we have a hyperrectangle $[\underline{\mathbf{x}}_{t+T}, \overline{\mathbf{x}}_{t+T}]$ that is guaranteed to contain the true reachable set $\mathcal{R}_T(\mathcal{X}_t)$.

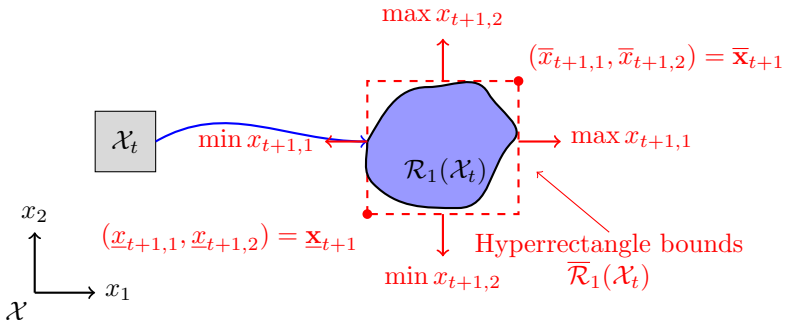

Figure 4: Illustration of hyperrectangle bounds (dashed red) for a 1-step reachable set. The true reachable set $\mathcal{R}_1(\mathcal{X}_t)$ (blue) has a non-convex shape and is fully contained within the hyperrectangle bounds, which form a rectangular over-approximation. The bounds are defined by the extreme points $\underline{\mathbf{x}}_{t+1}$ and $\overline{\mathbf{x}}_{t+1}$ (red dots), which are computed by solving optimization problems for the minimum and maximum values in each dimension.

7 Bounds in Matrix Form

Having computed the hyperrectangle bounds in Section 6, we can express these bounds more compactly in matrix form. The upper and lower bounds $\overline{\mathbf{x}}_{t+T}$ and $\underline{\mathbf{x}}_{t+T}$ together define an axis-aligned hyperrectangle that contains the reachable set $\mathcal{R}_T(\mathcal{X}_t)$.

This containment constraint $\underline{\mathbf{x}}_{t+T} \leq \mathbf{x}_{t+T} \leq \overline{\mathbf{x}}_{t+T}$ can be written as a system of linear inequalities in matrix form:

$$\begin{bmatrix} I \\ -I \end{bmatrix} \mathbf{x}_{t+T} \leq \begin{bmatrix} \overline{\mathbf{x}}_{t+T} \\ -\underline{\mathbf{x}}_{t+T} \end{bmatrix} \tag{27}$$

This elegant representation combines both the upper and lower bounds into a single inequality. The upper part expresses $\mathbf{x}_{t+T} \leq \overline{\mathbf{x}}_{t+T}$ (the upper bounds from our maximization problems), while the lower part expresses $-\mathbf{x}_{t+T} \leq -\underline{\mathbf{x}}_{t+T}$, which is equivalent to $\mathbf{x}_{t+T} \geq \underline{\mathbf{x}}_{t+T}$ (the lower bounds from our minimization problems).

These matrix bounds can be used directly in optimization or verification algorithms, providing a concise way to represent the hyperrectangle that over-approximates the reachable set. In fact, many neural network verification software implementations, such as auto-LiRPA[1] and JAX-Verify[2], accept a matrix specification, allowing you to plug $\begin{bmatrix} I \\ -I \end{bmatrix}$ directly into these tools and get the complete hyperrectangle bounds in a single function call.

8 More General Bounds

If we choose $\mathbf{c}$ vectors to point in different directions (not just axis-aligned), we can get more general polytope bounds. Let's denote these direction vectors as $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m$. For each direction $\mathbf{c}_i$, we solve an optimization problem:

$$d_i = \max_{\mathbf{x}_t \in \mathcal{X}_t} \mathbf{c}_i^T \mathbf{x}_{t+T} \quad \text{s.t.} \quad \mathbf{x}_{t+T} = f^T(\mathbf{x}_t, \pi) \tag{28}$$

We can then stack the vectors $\mathbf{c}_i$ to form a matrix $C = [\mathbf{c}_1^T; \mathbf{c}_2^T; \ldots; \mathbf{c}_m^T]$, and collect the optimization results to form a vector $\mathbf{d} = [d_1; d_2; \ldots; d_m]$. This gives us the polytope bounds:

$$\overline{\mathcal{R}}_T(\mathcal{X}_t) = \{\mathbf{x}_{t+T} \mid C\mathbf{x}_{t+T} \leq \mathbf{d}\} \supseteq \mathcal{R}_T(\mathcal{X}_t) \tag{29}$$

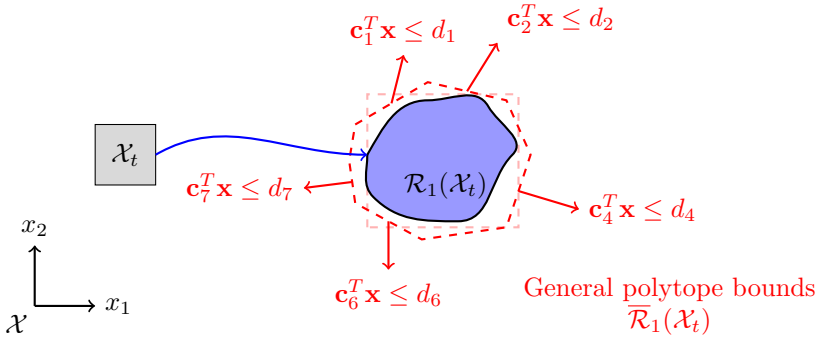

Figure 5: Illustration of general polytope bounds (red dashed octagon) compared to hyperrectangle bounds (faded red dashed rectangle). By using different objective vectors $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_7$ that are normal to the polytope faces, we can obtain tighter over-approximations of the reachable set $\mathcal{R}_1(\mathcal{X}_t)$ (blue). Each face of the polytope corresponds to a halfspace constraint of the form $\mathbf{c}_i^T\mathbf{x} \leq d_i$, where the $\mathbf{c}_i$ vectors point in the outward normal direction.

Using more general polytope bounds with carefully chosen direction vectors $\mathbf{c}$ allows for tighter approximations of the reachable set compared to axis-aligned hyperrectangles. Each face of the polytope corresponds to a halfspace constraint $\mathbf{c}_i^T\mathbf{x} \leq d_i$, where $\mathbf{c}_i$ is the outward normal vector to that face. The resulting polytope can more closely conform to the shape of the true reachable set, potentially reducing conservatism in verification tasks.

[1] `https://github.com/KaidiXu/auto_LiRPA`
[2] `https://github.com/google-deepmind/jax-verify`

References

[1] Katz, G., Barrett, C., Dill, D.L., Julian, K., & Kochenderfer, M.J. (2017). *Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks.* In Computer Aided Verification. CAV 2017. Lecture Notes in Computer Science, vol 10426, pp. 97–117.