Backward Reachability Analysis of Neural Feedback Systems

EECE/CS 7268: Verifiable Machine Learning

Michael Everett

1 Introduction to Backward Reachability

In our exploration of neural network verification, understanding which input states can lead to specific output states is crucial. Last week, we introduced the forward reachable set:

$$\mathcal{R}_1(\mathcal{X}_t) = \{f(\mathbf{x}_t, \pi) \mid \forall \mathbf{x}_t \in \mathcal{X}_t\} \tag{1}$$

where we needed to specify $\mathcal{X}_t$ (the input state set). Instead of determining "starting from $\mathcal{X}_t$, what states could the system enter in the future," we now want to find the set of states that lead into $\mathcal{X}_t$.

2 Backward Projection Sets

Let's reframe our analysis from a backward perspective to find the set of states that lead into $\mathcal{X}_t$.

Definition 1 (Backward Projection Set)**.** The one-step backward projection set is defined as:

$$\mathcal{P}_1(\mathcal{X}_t) = \{\mathbf{x}_{t-1} \mid f(\mathbf{x}_{t-1}, \pi) \in \mathcal{X}_t\} \tag{2}$$

This represents the states from which the system will enter $\mathcal{X}_t$ in exactly 1 step.

Definition 2 (T-Step Backward Projection Set)**.** The T-step backward projection set is defined as:

$$\mathcal{P}_T(\mathcal{X}_t) = \left\{\mathbf{x}_{t-T} \mid f^T(\mathbf{x}_{t-T}, \pi) \in \mathcal{X}_t\right\} \tag{3}$$

This represents the states from which the system will enter $\mathcal{X}_t$ in exactly T steps.

These concepts may be familiar from your previous studies of functions in mathematics. You may have discussed computing the image of a set under a function mapping (i.e., where does the function take the input set?). In our forward reachability analysis, we were computing the image of the initial state set under the closed-loop system dynamics. Now, with backward reachability, we're computing the pre-image (i.e., finding the set of initial states that lead to our target output set). Figure 1 illustrates this relationship between a function's image and pre-image.

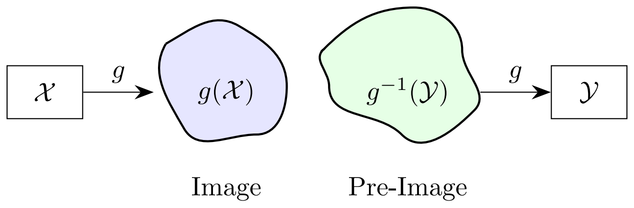

Figure 1: Relationship between function mapping, image, and pre-image

3 Computing Backward Reachable Sets

Just as it was difficult to compute the forward reachable set $\mathcal{R}_1(\mathcal{X}_t)$ exactly for neural networks or neural feedback loops, it is similarly challenging to compute the backward projection set $\mathcal{P}_1(\mathcal{X}_t)$ exactly. The non-linearities in neural networks create computational challenges in both the forward and backward directions. Therefore, as with forward reachability, we will focus on computing bounds when performing backward reachability analysis.

In forward reachability analysis, we were mainly interested in outer bounds and over-approximations to ensure safety guarantees. However, in backward reachability analysis, both outer bounds and inner bounds have practical meaning, depending on the problem at hand. Outer bounds help us identify areas that might lead to target states, while inner bounds give us guaranteed regions that definitely lead to target states.

3.1 Outer and Inner Bounds

Figure 2 illustrates the relationship between the exact backward projection set $\mathcal{P}_1(\mathcal{X}_t)$ (shown in purple) and its approximations. The outer approximation $\bar{\mathcal{P}}_1(\mathcal{X}_t)$ (shown in red) completely contains the true set but may include additional states that don't actually project into $\mathcal{X}_t$. The inner approximation $\underline{\mathcal{P}}_1(\mathcal{X}_t)$ (shown in green) is fully contained within the true set, representing states that are guaranteed to project into $\mathcal{X}_t$. The relationship between these sets can be expressed as $\underline{\mathcal{P}}_1(\mathcal{X}_t) \subseteq \mathcal{P}_1(\mathcal{X}_t) \subseteq \bar{\mathcal{P}}_1(\mathcal{X}_t)$.

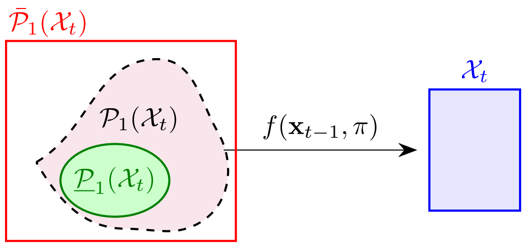
$\bar{\mathcal{P}}_1(\mathcal{X}_t)$
$\mathcal{P}_1(\mathcal{X}_t)$
$\underline{\mathcal{P}}_1(\mathcal{X}_t)$
$f(\mathbf{x}_{t-1}, \pi)$
$\mathcal{X}_t$

Figure 2: Inner and outer bounds on the backward reachable set

When would we want under vs. over approximations?

- If the target set $\mathcal{X}_t$ is a goal set, we want the inner approximation (under-approximation) $\underline{\mathcal{P}}_1(\mathcal{X}_t)$ so that we are sure the true system will reach the goal from any state in this set. If our under-approximation is too small, then we will be conservative (there may be states that we did not include that do reach the goal) but sound (there will not be any states in our set for which the true system will not reach the goal).
- In contrast, if the target set $\mathcal{X}_t$ is an obstacle or avoid set, we want the outer approximation (over-approximation) $\bar{\mathcal{P}}_1(\mathcal{X}_t)$ so that we are sure the true system will not enter the obstacle from any state outside this set. If our over-approximation is too big, then we will be conservative (there may be states that do not collide with the obstacle that we unnecessarily included in our set) but sound (there will not be any states outside our set for which the true system will collide with the obstacle).

4 Challenges with Inverting Neural Networks

For "classical" control systems, backward reachability and forward reachability are roughly equivalent problems if f^{-1} can be obtained. With neural networks, computing f^{-1} presents significant challenges. For example,

- Forward: $x \in [-1, 1] \Rightarrow \text{ReLU}(x) \in [0, 1]$ (finite interval, which can be propagated to other neurons appropriately)
- Backward: $\text{ReLU}(x) \in [0, 1] \Rightarrow x \in (-\infty, 1]$ (while correct for this one neuron, propagating this infinite interval back further would lead to vacuous results).

There are simple workarounds for this specific issue (e.g., using using other activation functions, such as Leaky ReLU, sigmoid). But other issues in inverting f can arise simply from the dimensions of weight matrices (e.g., if a weight matrix is singular or has more columns than rows).

There are some papers on neural network architectures that are designed to be invertible by construction [1]. There is also evidence that many neural networks trained in practice may actually be invertible despite the above properties that rule out the general case [2].

5 Alternative Strategy for Backward Reachability

We need a different strategy than inverting f. It would be great if we could use tools from forward reachability, but those require specifying input bounds, which is exactly what we're trying to compute here.

Remark 1. Key idea: Start with (very) conservative input bounds that are known to contain $\mathcal{P}_1(\mathcal{X}_t)$, then iteratively refine these bounds.

6 One-Step Backward Reachable Sets

Definition 3 (One-Step Backward Reachable Set)**.** One practical way of implementing Remark 1 is to define the backward reachable (BR) set:

$$\mathcal{BR}_1(\mathcal{X}_t) = \{\mathbf{x}_{t-1} \mid \exists \mathbf{u}_{t-1} \text{ such that } p(\mathbf{x}_{t-1}, \mathbf{u}_{t-1}) \in \mathcal{X}_t\}, \tag{4}$$

for a convex set of control limits $\mathcal{U}$. This set $\mathcal{U}$ would often be known in practice anyway, e.g., to prevent a robot from outputting a huge torque that would damage the motors.

In words, this set includes all states such that – for some control in the control limits – the system will move into $\mathcal{X}_t$ in one timestep. This set ignores the specific policy π and instead considers any possible π. We are creating a sound abstraction: if the property holds for any possible π, then it will certainly hold for our specific π as well.

Proposition 1. $\mathcal{BR}_1(\mathcal{X}_t) \supseteq \mathcal{P}_1(\mathcal{X}_t)$

7 Computing $\mathcal{BR}_1(\mathcal{X}_t)$ for Linear Systems

If the dynamics p are linear, we can find convex outer bounds on $\mathcal{BR}_1(\mathcal{X}_t)$ by solving linear programs (LPs):

$$\min_{\mathbf{x}_{t-1}, \mathbf{u}_{t-1}} \quad \mathbf{c}^T \mathbf{x}_{t-1} \tag{5}$$

$$\text{s.t.} \quad A\mathbf{x}_{t-1} + B\mathbf{u}_{t-1} \in \mathcal{X}_t \tag{6}$$

$$\mathbf{u}_{t-1} \in \mathcal{U} \tag{7}$$

For example, solving with $\mathbf{c}$ as each row of $\begin{bmatrix} I \\ -I \end{bmatrix}$ will produce a vector $\begin{bmatrix} \mathbf{g} \\ -\mathbf{h} \end{bmatrix}$, which forms hyper-rectangular/interval outer bounds on $\mathcal{BR}_1(\mathcal{X}_t)$:

$$\bar{\mathcal{BR}}_1(\mathcal{X}_t) = \{\mathbf{x}_{t-1} \mid \mathbf{g} \leq \mathbf{x}_{t-1} \leq \mathbf{h}\}. \tag{8}$$

8 Tightening the bounds for a specific policy π

Now that we have $\bar{\mathcal{BR}}_1(\mathcal{X}_t)$, we have an "input set" over which we can perform a relaxation of the system in the forward direction. For example, we can use CROWN on the closed-loop dynamics $\mathbf{x}_t = f(\mathbf{x}_{t-1}; \pi)$

over the domain $\bar{\mathcal{BR}}_1(\mathcal{X}_t)$ to get terms $\underline{M}, \overline{M}, \underline{\mathbf{n}}, \overline{\mathbf{n}}$. Then we can form an outer-bound $\bar{\mathcal{P}}_1(\mathcal{X}_t)$ on the backprojection set $\mathcal{P}_1(\mathcal{X}_t)$:

$$\bar{\mathcal{P}}_1(\mathcal{X}_t) = \{\mathbf{x}_{t-1} \in \bar{\mathcal{BR}}_1(\mathcal{X}_t) \mid \underline{M}\mathbf{x}_{t-1} + \underline{\mathbf{n}} \leq \mathbf{x}_t \leq \overline{M}\mathbf{x}_{t-1} + \overline{\mathbf{n}}\}. \tag{9}$$

Whereas $\bar{\mathcal{BR}}_1(\mathcal{X}_t)$ is very loose because it considers any possible π, this new set is specific to our policy π. If the new constraints based on $\underline{M}, \overline{M}, \underline{\mathbf{n}}, \overline{\mathbf{n}}$ are active (i.e., they are tighter than the constraints that define $\bar{\mathcal{BR}}_1(\mathcal{X}_t)$), then we have improved bounds. In practice, these new constraints are not always active, particularly if $\bar{\mathcal{BR}}_1(\mathcal{X}_t)$ is very big since that usually causes the affine bounds computed by CROWN to be quite loose.

9 Iterative Refinement Process

To summarize, we ultimately want to get an outer bound on $\mathcal{P}_1(\mathcal{X}_t)$. We started with $\bar{\mathcal{BR}}_1(\mathcal{X}_t)$ as a (very loose) outer bound on $\mathcal{P}_1(\mathcal{X}_t)$. We used $\bar{\mathcal{BR}}_1(\mathcal{X}_t)$ to obtain a (potentially) even tighter bound, $\bar{\mathcal{P}}_1(\mathcal{X}_t)$, such that:

$$\underbrace{\bar{\mathcal{BR}}_1(\mathcal{X}_t)}_{\text{old bound}} \supseteq \underbrace{\bar{\mathcal{P}}_1(\mathcal{X}_t)}_{\text{new bound}} \supseteq \mathcal{P}_1(\mathcal{X}_t). \tag{10}$$

If $\bar{\mathcal{BR}}_1(\mathcal{X}_t) \supset \bar{\mathcal{P}}_1(\mathcal{X}_t)$ (strict superset), we can do another iteration of refinement. That is, let's use $\bar{\mathcal{P}}_1(\mathcal{X}_t)$ to compute a tighter $\bar{\mathcal{P}}'_1(\mathcal{X}_t)$ such that:

$$\bar{\mathcal{BR}}_1(\mathcal{X}_t) \supseteq \underbrace{\bar{\mathcal{P}}_1(\mathcal{X}_t)}_{\text{old bound}} \supseteq \underbrace{\bar{\mathcal{P}}'_1(\mathcal{X}_t)}_{\text{new bound}} \supseteq \mathcal{P}_1(\mathcal{X}_t). \tag{11}$$

This iterative refinement process can continue until the set stops tightening, or we reach a satisfactory approximation, or some other desired stopping condition.

10 Advanced Refinement Strategies

The strategy described so far is sound but often leads to loose bounds on $\mathcal{P}_1(\mathcal{X}_t)$. Some strategies to get even tighter bounds include:

- Branch-and-bound: Split $\bar{\mathcal{BR}}_1$ and/or $\mathcal{X}_t$ into sub-regions and compute $\bar{\mathcal{P}}_1$ per sub-region, then combine appropriately [3]
- Tighter bound propagation: Naively propagating $\bar{\mathcal{BR}}_1$ forward through the closed-loop dynamics with CROWN temporarily ignores knowledge of the target set, $\mathcal{X}_t$. There are ways to modify the linear relaxation procedure to keep the target set constraint [4]
- When computing backprojection sets for multiple timesteps, tradeoffs between symbolic and concrete approaches appear just like in forward reachability
- Other ideas described in papers such as [5, 6, 7, 8]

11 Conclusion

Backward reachability analysis for neural networks presents unique challenges due to the non-invertibility of neural network functions. The approach presented here involves:

1. Starting with conservative bounds that contain the backprojection set
2. Using forward reachability tools and techniques like CROWN to establish tighter bounds
3. Iteratively refining these bounds to better approximate the true backprojection set

This methodology provides a practical way to compute backward reachability for verification of neural network-controlled systems.

References

[1] Lynton Ardizzone et al. "Analyzing inverse problems with invertible neural networks". In: *arXiv preprint arXiv:1808.04730* (2018).

[2] Joseph A Vincent and Mac Schwager. "Reachable polyhedral marching (rpm): An exact analysis tool for deep-learned control systems". In: *IEEE Transactions on Neural Networks and Learning Systems* (2025).

[3] Nicholas Rober. *BReach-LP: a Framework for Backward Reachability Analysis of Neural Feedback Loops.* Massachusetts Institute of Technology, 2023.

[4] Suhas Kotha et al. "Provably bounding neural network preimages". In: *Advances in Neural Information Processing Systems* 36 (2023), pp. 80270–80290.

[5] Anton Björklund, Mykola Zaitsev, and Marta Kwiatkowska. "Efficient Preimage Approximation for Neural Network Certification". In: *arXiv preprint arXiv:2505.22798* (2025).

[6] Xiyue Zhang et al. "Premap: A unifying preimage approximation framework for neural networks". In: *Journal of Machine Learning Research* 26.133 (2025), pp. 1–44.

[7] Chelsea Sidrane and Jana Tumova. "BURNS: Backward Underapproximate Reachability for Neural-Feedback-Loop Systems". In: *arXiv preprint arXiv:2505.03643* (2025).

[8] Michael Everett, Rudy Bunel, and Shayegan Omidshafiei. "Drip: Domain refinement iteration with polytopes for backward reachability analysis of neural feedback loops". In: *IEEE Control Systems Letters* 7 (2023), pp. 1622–1627.