Quantifying Safety of Learning-based Self-Driving Control Using Almost-Barrier Functions: Supplementary Materials

Zhizhen Qin Tsui-Wei Weng Sicun Gao

I. Pseudocode

Algorithm 1 shows the pseudocodes for data collection and the initial training of the control barrier function. Algorithm 2 illustrates the step for certifying a trained barrier function, and finding violations on the boundary, which can be further used from retraining.

II. Policy Learning

We use the following reward function to train the neural controller with Soft Actor Critic:

$$r_t = \begin{cases} -|d_t^s| - |d_t^p| - |d_t^a|, & s_t \text{ is not terminal} \\ 100, & s_t \text{ is goal} \\ -100, & s_t \text{ is out of the track} \end{cases} \quad (1)$$

Where d_t^s, d_t^p, d_t^a represent speed error, distance error and cross-angle error, respectively.

III. Learning Low-Dimensional Barriers under Partial Observability

When the barrier function is over full state dimensions, evaluating the loss function is straightforward: we can set the environment to the desired states and simulate one time step from neural controller to obtain the dynamics. For the dynamic model with lateral slip, we observe the need of training a lower-dimensional barrier function, over partially observed states. This is due to both scalability and feasibility. When a sharp corner is encountered with high speed, it is necessary for the vehicle to decelerate in order not to deviate from the track and crash. Fig. 1 shows the trajectories of the vehicle in three dimensions are already with a highly non-convex shape. It is hard to train a neural network to precisely capture this shape. In addition, with high-dimensions the certification procedures will suffer from curse of dimensionality. Thus, it is important to design methods for capturing the barrier in low-dimensional projections of the states.

The main difficulty for training low-dimensional barriers under partial observability is the lack of full-state information of the observations that are not seen in the collected trajectories, for approximating the dynamics and Lie derivatives. By further examining longitudinal speed, lateral speed and raw rate in Fig.1, we observe that at the boundaries where distance and angle errors are large, the other dimensions vary in a very small range. Thus, we can use nearest neighbors to find the dynamics in the unobserved dimensions. Formally, suppose the full states of the environment is $x \in \mathbb{R}^n$ and we aim to train a barrier with partial observation $o \in \mathbb{R}^m$, where $m < n$. After collecting trajectories with full states we select the observation dimensions we want to train a barrier function on, and form an observation set O_t in $\mathbb{R}^m$. Then, we can learn the system dynamics for a boundary state in the observation space $\mathbb{R}^m$ by finding its nearest neighbor in O_t. For the dynamic model, we can find the nearest neighbor's corresponding full state in $\mathbb{R}^n$ and replace the observation dimensions with the one from the boundary observation. We then use the synthesized full state to simulate the system dynamics of the boundary state. In the black box model, we use the dynamics of nearest neighbor states to calculate the Lie derivatives of the boundary states.

In Fig. 2 we plot the vector field of the the dynamics model. The left figure shows the dynamics of the points that are close to the collected trajectories, and the right figure shows the dynamics of the whole grid (note that we do not have to find the dynamics for the whole grid: here we are just computing it to show the calculated dynamics). As can be shown, at the boundary, the arrows tend to converge to the center, implying the barrier behavior.

IV. Estimating Range of Dynamics near Samples

To find the interval bounds on the Lie derivatives, we need to bound the variation of the dynamics of the system in the neighborhood of a sampled state. Let a be a sampled sate, and we write the hyperbox around a as $[a]_\delta$. Using interval arithmetic, we have

$$L_f B([a]_\delta) \subseteq \nabla B([a]_\delta) \circ f([a]_\delta)$$

where $\circ$ is interval dot product. First, we estimate the value of $f(a)$ through finite difference over time $f(x) \approx (x(t+\Delta t) - x(t))/\Delta t$ evaluated at $x(t) = a$. where we assume $\ddot{x}(t+\lambda\Delta t)\Delta t$ is tiny, $\lambda \in [0,1]$. That is, we can assume $\|\ddot{x}(t+\lambda\Delta t)\| \leq L$ and bloat the range around $f(a)$ negligibly by $L\Delta t$. Then consider an arbitrary state $a+p$ in the neighborhood of a:

$$f(a+p) = f(a) + J_f(a)p + \frac{1}{2}p^T H_f(a+\lambda p)p$$

where J_f and H_f are the Jacobian and Hessian-tensor of f (f is vector-valued). We use finite differences to estimate the Jacobian. Writing $f = (f_1, ..., f_n)$, with $\delta \to 0$, we have

$$\left.\frac{\partial f_j}{\partial x_i}\right|_{x=a} \approx \frac{f_j(a+\delta e_i) - f_j(a)}{\delta}$$

where e_i is the unit vector in the x_i coordinate. Thus by taking n (the number of dimensions) samples of $f(x+\delta e_i)$, we can get the estimate of the Jacobian of f, $J_f(a)$ (each sample used to estimate all functions in f), where e_i is the

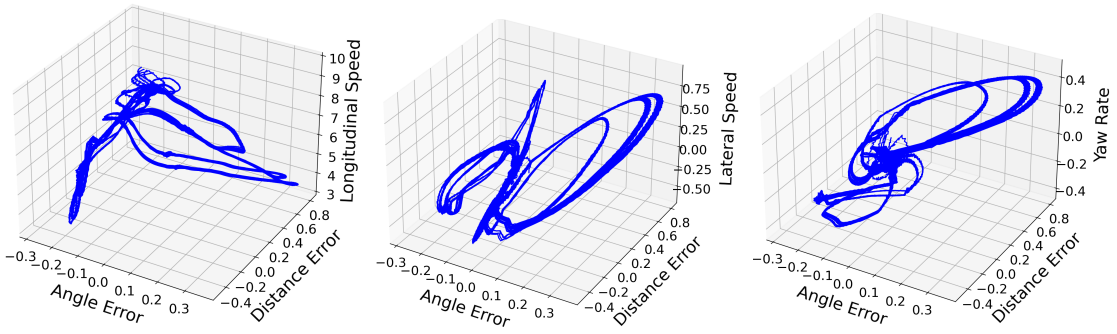

Fig. 1: Trajectories of neural controller on dynamic model. Left: angle error, distance error and lateral speed. Middle: angle error, distance error and lateral speed. Right: angle error, distance error and lateral speed. Middle: angle error, distance error and yaw rate.

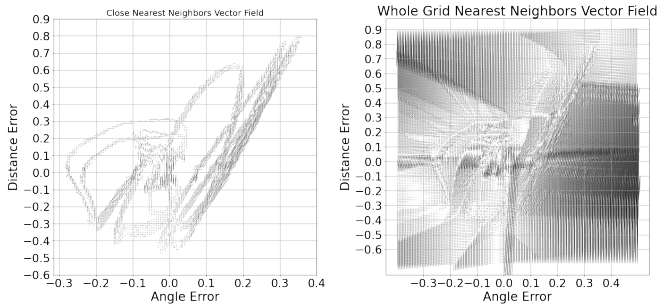

Fig. 2: Vector field of the system dynamics calculated with nearest neighbor approach for the dynamic model.

Algorithm 1 Learning Neural Barrier Functions

Input: Control policy π, unsafe set size N_u, number of nearest neighbors k, number of epochs K, barrier function learning rate α, weight parameters for loss function w_s, w_u, w_l, γ
Run the neural control policy to collect N_s safe samples X_s, together with the dynamics on these safe samples, which are the differences between adjacent states x' and x for each $x \in X_s$
Initialize unsafe set $X_u = \varnothing$
while $|X_u| < N_u$ **do**
    Sample a batch of candidate observations X_c uniformly randomly from the space
    Initialize to-be-removed observation set X_r
    for every $x_c \in X_c$ **do**
        Neighbors $X_n = k\text{NN}(x_c, X_s \cup X_u \cup X_c)$
        if $|X_n \cap X_s| > k/2$ **then**
            $X_r \leftarrow X_r \cup \{x_c\}$
        end if
    end for
    $X_u \leftarrow X_u \cup \left(X_c \setminus X_r\right)$
end while
Initialize barrier function network B_θ
for episodes $= 1, \ldots, K$ **do**
    Sample mini-batches of size n from $X_s \cup X_u$
    $\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta)$
end for

unit vector in the x_i coordinate, and again assuming that the second-order deviations are negligible by slight bloating of the estimated range. Note that each dimension of sample $f(x)$ requires 2 states to compute the dynamics, so overall we need $2n$ state samples for bounding $f([a]_\delta)$. Also note that to cover a hyperbox we need to consider the gradient direction for the most rapid change in it.

When we learn barriers on partial state dimensions, the range in unobserved dimensions implicitly affects the bounds on $f(a)$. For a full state x, we partition it and write the observation as a and unobserved dimensions as y. To get the dynamics bounds, besides $[a]_\delta$, we also need to consider a hypterbox around y, $[y]_\beta$. To do so, when finding dynamics in the observation space, after using kNN to find the full state,

Algorithm 2 Certifying Neural Barrier Functions

Input: Barrier function B
Initialize certified safe set X_{cs}, unsafe set X_{cu}, boundary set with certified lie derivative requirement X_{cb}, and boundary set that violates the requirement X_{vb}
for each point x in the grid **do**
    Calculate lower and upper bounds of barrier function $B_L(x)$ and $B_U(x)$ within an 2δ box
    if $B_L(x) > 0$ **then** $X_{cs} \leftarrow X_{cs} \cup \{x\}$ ▷ safe
    else if $B_U(x) < 0$ **then** $X_{cu} \leftarrow X_{cu} \cup \{x\}$ ▷ unsafe
    else
        Calculate lower and upper bounds of barrier function derivatives $d_L \leftarrow (\frac{\partial B(x)}{\partial x})_L$ and $d_U \leftarrow (\frac{\partial B(x)}{\partial x})_U$
        Calculate lower and upper bounds of dynamics $f_L(x)$ and $f_U(x)$ (abbreviated below as f_L, f_U)
        Initialize lie derivative lower bound $\left[\mathsf{L}_f B(x)\right]_L$ as $l \leftarrow 0$
        for each dimension i of the observation space **do**
            $l \leftarrow l + \min\left[d_{U,i} f_{U,i},\ d_{U,i} f_{L,i},\ d_{L,i} f_{U,i},\ d_{U,i} f_{U,i}\right]$
        end for
        if $l > 0$ **then**
            $X_{cb} \leftarrow X_{cb} \cup \{x\}$ ▷ certified boundary
        else
            $X_{vb} \leftarrow X_{vb} \cup \{x\}$ ▷ boundary violation
        end if
    end if
end for

we sample $\delta e_{a,i}$ and $\beta e_{y,i}$ for each coordinate, to find the dynamics bounds $f([a]_\delta, [y]_\beta)$. At deployment, we need to verify that a new state x''s unobserved dimensions y' must be within the corresponding $[y]_\beta$ that is used during certification.

V. Finding Boundary Counterexamples for Retraining

We leverage the verification bounds in the main paper to first find the violated observations: for each observation x, it is a violation if (1) $B_L(x) < 0$ and $B_U(x) > 0$, and (2) $\mathsf{L}_f B(x) < 0$. A violation state can be found with the following procedures. First, we calcuate the interval bounds of barrier function values B_U and B_L for states in the state space, and those with $B_U > 0$ and $B_L < 0$ are identified with boundary states x_b. We then calculate for the lie derivatives of the boundary states, which consists of the bounds of barrier function network derivatives and the bounds of dynamics. The former can be found using the methods in network robustness analysis. We can find dynamics bounds using the methods described in the previous section. With these information, we are able to compute the upper and lower bounds of $\mathsf{L}_f B(x_b)$ in an 2δ box for all boundary states, and identify the ones with lower bounds $\left[\mathsf{L}_f B(x_b)\right]_L < 0$ as violations. We apply the kNN approach for labeling the violation states as well.

For each observation, find k nearest labels from $X_s \cup X_u$. If more than ρk neighbors are from X_s or X_u ($0 < \rho < 1$), it is added into X_s or X_u, respectively. Otherwise the violation set remains the same. We can then retrain the neural barrier functions with the updated safe and unsafe sets by minimizing the loss function. We continue doing so until the number of violations is smaller than some threshold.

VI. Using the Learned Barrier Function for Safety Monitor

As the certification step illustrated, the learned barrier function may contain uncertified regions on its zero-level set, where the upper bound of the their lie derivatives is positive. It means that when the system reaches these states, it is possible that they will escape the safe region of the space. Since the violation region is explicitly computed, through reachability analysis we can construct runtime monitors that can alert the human operators for states that may reach the uncertified regions in the near future. Formally, an online monitor can be defined by a superlevel set of the barrier function $M = \{x \in X : B(x) = c\}$ for some $c > 0$, where X is the state space and B is the learned barrier function, so that M is contained in the safe set. The constant c can be chosen based in the predictive time horizon T. Write the discretized trajectories of the controlled system as $\tau^T(x_0) = \{x(0), x(1), \cdots, x(T)\}$ with initial state $x_0 = x(0)$. Over the monitor set M, we aim to identify safe regions M_s^T and unsafe regions M_u^T, such that $M \subseteq M_s^T \cup M_u^T$ and for all $x_s \in M_s^T$, $\tau(x_s)^T \subseteq X_s$, i.e. any trajectory starting from M_s stays within the safe set for the entire duration of T, while the states in M_u may reach the uncertified region on the safety barrier within time T. Although in general, characterizing M_s and M_u requires backward reachability computation, in the case of path-tracking that we only need to consider the maximum curvature case to know the worst case trajectories of the vehicles. Specifically, we can construct paths with a specified curvature, and check for trajectories starting from the monitor set M. Since we have verified Lie derivative conditions for certified region of the barrier, we only need to check for uncertified regions. Denote uncertified barrier

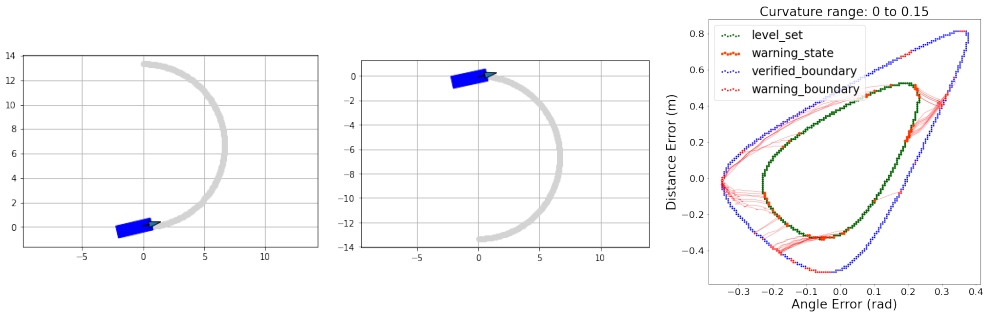

Fig. 3: **Left and Middle**: evaluation paths with curvature of 0.15, curving to the left and right. **Right**: Illustration of the safety monitor on a level set M ($c = 0.07$) inside the safe region (inner curve), the red segments show M_u^T, states that could potentially cross the safety boundary in high curvature environments, while states in the green regions, M_s^T, are guaranteed safe within the time window, with $T = 50$.

region as $x_{ub} \in X_{ub}$, and the hyperbox around state x as $[x]_\delta$. For all $x_M \in M$, we simulate T time steps to get trajectory $\tau(x_M)^T$. If $x \in [x_{ub}]_\delta$ for any $x \in \tau(x_M)^T$, it is classified into unsafe monitor region M_u^T; otherwise, it is classified into safe region M_s^T. For low-dimensional barriers, we fill the additional dimensions using the same kNN technique described in Section III of this paper. The first two figures in Fig. 3 illustrate the evaluation paths for a maximum curvature of 0.15, and the third one shows the monitor results for this curvature on dynamic vehicle model, with $T = 50$ and $c = 0.07$.

VII. Hyper-parameters and Details of Experiments

In this section we provide hyper-parameters used in experiments. As mentioned in the main paper, the control policies are represented by two-layer fully connected neural networks, with 128 hidden nodes in each layer and ReLU activations. The neural network for the barrier function also has a single layer, with 256 hidden nodes and *tanh* activation function. The hyper-parameters for training the barrier functions is shown in Table I. The computation is performed on a PC with Intel i7-8700 and GTX 1080.

TABLE I: Hyper-parameters for barrier function training on three environments

	w_s	w_u	w_l	γ
Kinematic	15	50	10	100
Dynamic	15	10	10	100
TORCS	15	15	10	100