Synthesizing Barrier Certificates Using Neural Networks

Hengjun Zhao*
School of Computer and Information Science
Southwest University, Chongqing, China
zhaohj2016@swu.edu.cn

Xia Zeng†
School of Computer and Information Science
Southwest University, Chongqing, China
xzeng0712@swu.edu.cn

Taolue Chen‡
Department of Computer Science and Information Systems
Birkbeck, University of London
taolue@dcs.bbk.ac.uk

Zhiming Liu§
School of Computer and Information Science
Southwest University, Chongqing, China
zhimingliu88@swu.edu.cn

ABSTRACT

This paper presents an approach of safety verification based on neural networks for continuous dynamical systems which are modeled as a system of ordinary differential equations. We adopt the deductive verification methods based on barrier certificates. These are functions over the states of the dynamical system with certain constraints the existence of which entails the safety of the system under consideration. We propose to represent the barrier function by neural networks and provide a comprehensive synthesis framework. In particular, we devise a new type of activation functions, i.e., Bent-ReLU, for the neural networks; we provide sampling based approaches to generate training sets and formulate the loss functions for neural network training which can capture the essence of barrier certificate; we also present practical methods to check a learnt candidate barrier certificate against the criteria of barrier certificates as a formal guarantee. We implement our approaches via proof-of-concept experiments with encouraging results.

CCS CONCEPTS

• **General and reference** → **Verification**; • **Computer systems organization** → **Embedded and cyber-physical systems**; • **Computing methodologies** → **Machine learning**.

*Hengjun Zhao was supported partially by the National Natural Science Foundation of China (No. 61702425, 61572024, 61972385), "Fundamental Research Funds for the Central Universities" (SWU116079), and Basic Science and Frontier Technology Research Program of Chongqing (cstc2017jcyjAX0295).
†Xia Zeng is the corresponding author and is supported partially by the National Natural Science Foundation of China (No. 61902325), and "Fundamental Research Funds for the Central Universities" (SWU117058).
‡Taolue Chen is partially supported by Birkbeck BEI School Project (ARTEFACT), the National Natural Science Foundation of China (No. 61872340), Guangdong Science and Technology Department Grant (No. 2018B010107004), Natural Science Foundation of Guangdong Province, China (No. 2019A1515011689).
§Zhiming Liu was supported partially by the National Natural Science Foundation of China (No. 61672435, 61732019, 61811530327), and Capacity Development Grant of Southwest University (SWU116007).

HSCC '20, April 22–24, 2020, Sydney, NSW, Australia

ACM ISBN 978-1-4503-7018-9/20/04…$15.00
https://doi.org/10.1145/3365365.3382222

KEYWORDS

Barrier certificates, Neural networks, Continuous dynamical systems, Verification

ACM Reference Format:
Hengjun Zhao, Xia Zeng, Taolue Chen, and Zhiming Liu. 2020. Synthesizing Barrier Certificates Using Neural Networks. In *23rd ACM International Conference on Hybrid Systems: Computation and Control (HSCC '20), April 22–24, 2020, Sydney, NSW, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3365365.3382222

1 INTRODUCTION

This paper is concerned with verification of continuous dynamical systems which are specified by a system of differential ordinary equations (ODEs). They are an indispensable component of hybrid systems, which feature interacting discrete and continuous dynamics and play a foundation role in the modeling of cyber-physical systems. Indeed, a typical hybrid system can be viewed, in a large sense, as continuous dynamical systems over discrete modes which may jump from each other when certain transitions are triggered. Many of these systems, especially those applied in, e.g., aircrafts, automobiles, chemicals and nuclear power plants, are safety-critical systems, where safety refers to, in the most fundamental form, that the system cannot reach a dangerous or unwanted state.

Safety verification of continuous dynamical systems, i.e., to determine whether the underlying system is absent from a dangerous or unwanted state, is of paramount importance. However, due to its inherent complexity, it also poses a long-standing challenge. In a nutshell, there are generally two classes of methods of analysis and verification of continuous dynamical systems. One is via directly computing sets of reachable states including the computation of the over—or in some cases under—approximations thereof [4, 13, 14, 32, 45]. For some special families of ODEs (especially when their vector fields are linear), one could also use symbolic approaches reducing to, for instance o-minimal theory [15, 16, 22]. However, as generally ODEs do not have closed form solutions and the numerical methods are usually computation intensive, the scalability of these approaches poses a severe limitation.

Alternatively, deductive methods have been studied and successfully applied in practice [26, 31, 33]. They are rooted at classic program verification methods (such as Hoare logic [17]) and are essentially based on the notion of (inductive) invariants; typical examples include loop invariants, pre/post contracts for functions,

or even rely-guarantee contracts for concurrent programs. In the setting of continuous dynamical system, or hybrid system in general, because of the continuous dynamics, one usually considers differential invariants [27, 34]. Suppose one can discover such an invariant which can be shown to hold at all reachable states, but not at those unsafe states, then immediately the safety of the system under consideration can be asserted. The reason that this class of approaches may be more effective than the previous one lies in that an invariant is only a (rough) approximation of the reachable set and may be discovered with awareness of the safety specification and according to the ODEs, rather than their solutions. The challenge, however, is how to discover such an invariant. An recurring pattern for synthesizing invariants is to reduce to constraint solving problems. More concretely, one may first predefine a property template (linear or non-linear, depending on the property to be verified) and then encode the conditions of a useful invariant into some constraints on state variables and parameters which is followed by the endeavor to find out solutions to the constraints.

The method considered in the current paper falls into the second category. In particular, we follow a deductive method based on the notion of barrier certificate [35], which has attracted much attention lately. A barrier certificate is a function of states that divides the state space into two parts. All system trajectories starting from a given set of initial conditions fall into one side of the barrier certificate while the unsafe region locates on the other side. As common in deductive verification methods, there are generally two closely related questions which are the focus of the current research: (1) what is the appropriate form of barrier certificates? (2) how to efficiently (needless to say, automatically) synthesize a barrier certificate of a fixed shape. As one may expect, there is a delicate trade-off between the two questions: in general, expressive barrier certificates are powerful to handle involved system dynamics and unsafe sets, but usually bring computational intractability to synthesis. For instance, a substantial body of work along this line is to consider (especially for polynomial ODEs) barrier functions to be polynomial functions, and accordingly, the synthesis of barrier functions reduces, owing to rich theory and tools in real algebraic geometry, to polynomial optimization (for instance, Sum-Of-Squares methods and Semi-definite programming) or quantifier elimination. (A more detailed discussion of the line of work is deferred to the related work section.) Our main work can be summarized as providing a new trade-off of the expressiveness of barrier certificates and the synthesis tractability via neural networks and the accompanied data-driven invariant synthesis technique.

Contributions. In this paper, we propose a new class of barrier certificates for safety verification of continuous systems based on neural networks. The rationale is that, by the well-known universal approximation theorem [23], neural networks can approximate arbitrary functions, which would be an ideal candidate to represent barrier certificates. While enjoying the expressiveness of neural networks, as we will demonstrate later, the synthesis can also be performed in an efficient way which go through following 3 steps.

As the first step, we devise a new class of activation functions, Bent-ReLU, for barrier certificates. Salient properties of this class of functions include: (1) they are smooth, so automatically continuously differentiable as required by barrier certificates. (2) it has a similar shape as ReLU functions, so demonstrates piece-wise linear zero-level sets, which would facilitate training (cf. Section 3.1.1) and post-verification (cf. Section 4). In particular, the proposed new activation function can be both transformed equivalently into polynomial forms or approximated as piecewise linear forms, so can be used in other training scenarios where one can employ tools from polynomial optimization and SMT solvers for a rigorous analysis of the trained networks.

As the next step, we train the neural network where the application of standard learning algorithms requires to come up with a training set as well as the loss function. To generate training data, we devise various methods to sample points from respective regions including the initial and unsafe domains (cf. Section 3.2). For the loss function, we propose a new form which is able to encode the three conditions of barrier certificates. In a nutshell, the points which do not respect these conditions are penalized by the loss function, and when barrier certificates exist, the loss becomes 0. (In terms of standard supervised learning, this is to encode the empirical risks.) The details are given in Section 3.3. It is worth mentioning that, to increase the chance of successful learning of barrier certificates, we impose gradient controls of the learnt function, which should be treated as regularization in standard machine learning terms (cf. Section 3.4.2). As is well-known in machine learning, designing a suitable loss function is usually the key to a successful application. The importance of our work is not only to give a formulation of the loss functions which are effective in generation of barrier certificate, but also to provide a pattern for applications of similar data-driven approaches in other verification tasks. Once the training set and loss functions are available, we apply standard learning algorithms, supported by the open source deep learning framework PyTorch.

The third step is to check whether the learnt candidate barrier function from the previous steps, represented by a neural network, satisfies all the requirements of barrier certificates. This step is necessary because of the data-driven nature of the approach: the generalizability of neural networks likely, but does not provide a formal guarantee to, generate a barrier certificate. Cast as a special neural network verification problem, in the current work, the verification is tackled by both symbolic approaches which reduce to quantifier elimination and are implemented via the algebraic system Redlog [8] and Mathematica[1], and approaches based on piecewise linear approximation which are based on interval constraint propagation based SMT solver isat3. (Cf. Section 4.)

The main contributions of the paper are summarized as follows:

- We put forward a learning-based framework for synthesizing barrier certificates via neural networks training and verification. This is largely a data-driven approach, with little prior knowledge required, and enjoys great flexibility to effectively handle nonlinear (beyond polynomial) dynamics of ODEs.
- We instantiate the framework by proposing a new class of activation functions *Bent ReLU*. Moreover, we demonstrate how to generate training set, and to construct loss functions of neural networks to encode the constraints of barrier certificates, as well as regularization. We also provide practical methods to formally verify the learnt barrier certificates represented as neural networks.

[1] https://www.wolfram.com/mathematica/

- We carry out proof-of-concept case studies to showcase the efficacy of the approach (cf. Section 5).

To the best of our knowledge, this is the first paper to exploit neural networks as a representation of barrier certificates in continuous dynamical system verification, and to provide a comprehensive framework to support its synthesis.

1.1 Related Work

Our work is to formulate and synthesize neural network-type barrier certificates for the verification of continuous dynamical systems whereby the related work mainly concentrates on the method on barrier certificate generation and more generally, verification of neural networks and learning for program verification.

Barrier certificate generation. The seminal work of using barrier certificates in safety verification of hybrid systems was introduced by Prajna et al. [35, 36]. Following this line, studies have been focusing on various forms of barrier certificates, striving to strike a balance between expressiveness and tractability of synthesis. The endeavor includes: Kong et al. [20, 21] proposed a barrier certificate defined over an exponential condition for semi-algebraic hybrid systems. Dai et al. [6] discussed how to relax the condition of barrier certificates in a general way without losing their convexity. Sloth et al. [40] proposed a new barrier certificate for a special class of hybrid systems consisting of many interconnected subsystems. Zeng et al. [48] considered a Darboux-type barrier certificate which characterizes an algebraic curve that restricts the trajectories of the system from leaving it once they enter it. Platzer and Clarke [34] investigated differential invariants, which lift barrier certificates from defining invariant sub-level sets of differentiable functions to formulas which can feature Boolean combinations of equalities and inequalities, describing a richer class of continuous invariants. Sogokon et al. [41] modified the conditions on the derivative of barrier functions which are relaxed in a way analogous to vector Lyapunov functions so as to preserve the convexity of the search space and search for more general classes of barrier certificates at the same time. From a computational point of view, relaxation based methods provide much better efficiency at the cost of more conservative results. Among them, sum-of-squares (SOS) relaxation is the most popular one [6, 20, 36, 40, 47]. Instead of directly handling constraints with quantifiers, SOS relaxation converts them to more conservative constraints represented as either linear matrix inequalities (LMI) [20] or bilinear matrix inequalities (BMI) [36, 47]. In addition, to make the computation tractable, the degrees of the polynomial multipliers appearing in LMI or BMI must be bounded. Instead of SOS relaxation, our generation is based on learning method which merges all the constraints from sampling into an unconstrained optimization problem, in a similar vein to Xue et al.'s work [46]; moreover, most of existing barrier certificates are more or less formalized from conservative conditions to preserve convexity while our approach adopts the non-convex condition (strict barrier certificate).

Verification of neural networks. Learnt barrier certificates are subject to further verification which is related to formal verification of neural networks. This has attracted considerable research efforts in recent years, and the general problem is NP-hard [19]. A large body of research focuses on the robustness issue of neural networks. In particular, given an input subject to (adversarial) perturbations, one intends to determine whether the output of the neural network (e.g., the classification result) is invariant to these perturbations. Essentially, this is to estimate the output range of a given neural network on a compact set. There are now a wide range of methods including constraint-solving based approaches [19], optimization based approaches [11, 43, 44], abstract interpretation based approaches [24, 37], etc. Furthermore, recently work has been done for verification of control systems with neural network components [9, 10, 18, 42]. Note that the discussions are necessarily non-exhaustive as a reasonably detailed discussion requires an independent survey which is out of the scope of this paper.

Learning in program verification. Machine learning has been used extensively in program verification, in particular, for invariant synthesis. Data-driven synthesis techniques for invariant or interpolants have been studied lately [2, 3, 12, 25, 38, 39]. They are used to handle programs which manipulate complex data-structures, arrays, pointers, etc., or to reason over a complicated memory model and its semantics. In such a scenario, a black-box, data-driven guess-and-check approach, guided by a finite set of program configurations, has been shown to be advantageous. In general, they formulate the synthesis problem as a learning problem where off-the-shelf techniques (e.g., support vector machines, decision trees) are used. In contrast, our work is in the continuous regime and adopts neural networks, although it does share the same spirit.

2 PRELIMINARIES

Throughout this paper, $\mathbb{R}$ denotes the set of real numbers. For any natural number n, we write $[n] = \{1, \cdots, n\}$.

2.1 Constrained Continuous Dynamical System

We consider a continuous system given as a system of ordinary differential equations (ODEs).

A continuous dynamical system S is modeled by a finite number of first-order ordinary differential equations

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}),$$

where $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ is a column vector, $\dot{\mathbf{x}}$ denotes the derivative of $\mathbf{x}$ with respect to the time variable t, and $\mathbf{f}(\mathbf{x}) : \Omega \to \mathbb{R}^n$ is a vector field $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_n(\mathbf{x}))^T$ defined on an open subset of $\Omega \subseteq \mathbb{R}^n$. We assume that $\mathbf{f}$ satisfies the *local Lipschitz condition*, which ensures that given $\mathbf{x} = \mathbf{x}_0$, there exists a time $\mathcal{T} > 0$ and a unique time trajectory $\mathbf{x}(t) : [0, \mathcal{T}) \to \mathbb{R}^n$ such that $\mathbf{x}(0) = \mathbf{x}_0$, also denoted by $\mathbf{x}(t, \mathbf{x}_0)$.

We consider *constrained continuous dynamical systems* given by $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, where $\mathbf{f} : \Omega \to \mathbb{R}^n$ is the vector field, $X_D \subseteq \Omega$ is an evolution constraint (also called the state space, or system domain), $X_I \subseteq X_D$, $X_U \subseteq X_D$. For safety verification, we only consider trajectories initialized in X_I that are contained in X_D and check whether a trajectory exists that can reach an unsafe set X_U.

Definition 2.1 (Safety verification problem). A considered system $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$ is safe if $\forall \mathbf{x}_0 \in X_I$ and $\forall t \geq 0, \mathbf{x}(t, \mathbf{x}_0) \in X_D$ implies $\mathbf{x}(t, \mathbf{x}_0) \notin X_U$, i.e., the system never reaches X_U from X_I.

Given a continuously differentiable function $g : \mathbb{R}^n \to \mathbb{R}$, the *gradient* of g w.r.t. $\mathbf{x} = (x_1, \ldots, x_n)^T$ is defined by the column vector

$$\nabla g = \frac{\partial g}{\partial \mathbf{x}} = (\frac{\partial g}{\partial x_1}, \ldots, \frac{\partial g}{\partial x_n})^T.$$

Definition 2.2 (Lie derivative). Given a vector field $\mathbf{f} : \Omega \to \mathbb{R}^n$, the Lie derivative of a continuously differentiable function g w.r.t. $\mathbf{f}$, $\mathcal{L}_\mathbf{f} g : \Omega \to \mathbb{R}$, is defined as the inner product of $\mathbf{f}$ and ∇g:

$$\mathcal{L}_\mathbf{f} g(\mathbf{x}) = (\nabla g) \cdot \mathbf{f}(\mathbf{x}) = \sum_{i=1}^{n} \left(\frac{\partial g}{\partial x_i}(\mathbf{x}) \cdot f_i(\mathbf{x}) \right). \tag{1}$$

2.2 Barrier Certificate

Given a system Γ, a barrier certificate is a real-valued function $B(\mathbf{x})$ over the states of the system satisfying the condition that $B(\mathbf{x}) \leq 0$ for any reachable state $\mathbf{x}$ and $B(\mathbf{x}) > 0$ for any state in the unsafe set X_U. If such a function $B(\mathbf{x})$ exists, one can easily deduce that the reachable set of states and the unsafe set are disjoint, viz., the system can *not* reach a state in the unsafe set from the initial set.

There are several different formulations of barrier certificates without explicit reference to the solutions of the ODEs [6, 20, 36, 41]. In this paper, we will adopt what termed as *strict barrier certificate* [40] (aka. non-convex conditions) which imposes less conservative requirements of barrier certificate conditions and increases the chance of successful synthesis.

Theorem 2.3 (Nonconvex Condition). *Given a system* $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, *if there exists a continuously differentiable function* $B : X_D \to \mathbb{R}$ *s.t.*

(1) $B(\mathbf{x}) \leq 0$ *for* $\forall \mathbf{x} \in X_I$
(2) $B(\mathbf{x}) > 0$ *for* $\forall \mathbf{x} \in X_U$
(3) $\mathcal{L}_\mathbf{f} B(\mathbf{x}) < 0$ *for all* $\mathbf{x} \in X_D$ *s.t.* $B(\mathbf{x}) = 0$,

then the system Γ *is safe, and such* B *is a barrier certificate.*

2.3 Neural Networks

We introduce some basic concepts of (feed-forward artificial) neural networks (NNs). A typical NN consists of a number of interconnected neurons which are organized in a layered structure. Each neuron is a single processing element that responds to the weighted inputs received from other neurons.

In general, an NN represents a function h and can be represented as a composition of its layers. We normally reserve 0 and L for the indices of the input and output layers respectively, and all the other layers in between are hidden layers. Superscripts (l) are used to index layer l-specific variables. In particular, the layer l comprises neurons $n_i^{(l)}$ for $i \in [d^{(l)}]$, where $d^{(l)}$ is the dimension of the layer l. For a feed-forward NN, neuron $n_j^{(l-1)}$ of the layer $l-1$ is connected with neuron $n_i^{(l)}$ of layer l by a directed edge with weight $w_{ij}^{(l)} \in \mathbb{R}$, and each neuron $n_i^{(l)}$ of layer l is associated with a bias $b_i^{(l)} \in \mathbb{R}$.

The network is fed an input through its input layer, which is then propagated through the layers by successive application of linear calculations and activation functions until it reaches the output layer. More formally, each hidden layer $h^{(l)}$ is defined as $h^{(l)}(\mathbf{x}) = a(\mathbf{W}^{(l)}\mathbf{x} + \mathbf{b}^{(l)})$, where $\mathbf{W}^{(l)}$ is a weight matrix, $\mathbf{b}^{(l)}$ is an offset (aka. bias) vector, and a is an activation function. A more detailed formulation of the propagation is presented in Section 3.1.2.

The most common types of activation functions include ReLU (Rectified Linear Unit, i.e., $\max(0, x)$ for $x \in \mathbb{R}$), sigmoid, hyperbolic tangent, etc. Training of NN are usually through optimization, during which the parameters $\mathbf{W}$'s and $\mathbf{b}$'s are learned through an optimization algorithm (e.g., stochastic gradient descent) applied on the training set.

3 LEARNING BARRIER CERTIFICATES

In this section, we present our approach to learn barrier functions represented by neural networks. Given a system $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, the process of learning a barrier certificate for Γ consists of the following steps:

(1) Fix the NN structure that represents the candidate barrier function;
(2) Generate a set of training data from X_D, X_I, X_U;
(3) Encode the three conditions of barrier certificate in Theorem 2.3 into the *loss* of the NN on the generated training data;
(4) Train the NN until the loss stabilizes at 0, and the stabilized weights and biases give rise to a barrier certificate.

In the rest of this section we explain the details of the four steps.

3.1 The Structure of NN

Given a system $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, the basic structure of the NN used to learn a barrier certificate is as follows: it has one input layer, one output layer, and $L-1$ hidden layers; the number of neurons in the input layer is equal to the dimension of Γ, i.e. $d^{(0)} = n$; the number of neurons in the output layer is 1, i.e. $d^{(L)} = 1$. Essentially, such an NN represents a smooth scalar function defined on $\mathbb{R}^n$.

3.1.1 Activation Function. In this paper, we use activation functions in the following form:

$$a(x) = 0.5 \cdot x + \sqrt{0.25 \cdot x^2 + \epsilon} \tag{2}$$

with ϵ a small positive real number. Actually (2) gives a family of activation functions resembling ReLU, which we call *Bent ReLU* functions, and ϵ denotes how *bent* the function is (c.f. left part of Fig. 1 for an illustration). Bent ReLU functions facilitate our purpose of learning barrier certificate in the following aspects:

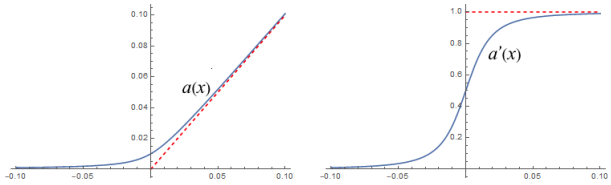

Figure 1: Bent ReLU and its derivative with $\epsilon = 0.0001$

- It is a *smooth* function stipulated by barrier certificates. The first-order derivative of Bent ReLU is

$$a'(x) = 0.5 + \frac{0.25 \cdot x}{\sqrt{0.25 \cdot x^2 + \epsilon}} \tag{3}$$

An illustration is given in the right part of Fig. 1.

- It provides the possibility of rigorously verifying the learned barrier in two different ways:
  - It is obvious from (2) and (3) that the relation between $a(x)$ and x, as well as the relation between $a'(x)$ and x can both be transformed to equivalent polynomial constraints. As a result, symbolic computation methods from first-order real algebra [5] can be employed.
  - When ϵ is sufficiently small, Bent ReLU can be approximated quite well by ReLU, which is piece-wise linear and can be well dealt with by SMT solvers. We will discuss the formal verification issue in more detail in Section 4.
- ReLU activation function is widely used in the machine learning community and learning NN with ReLU is intensively investigated. It is reasonable to assume that properties proved with ReLU will (at least to some extent) transfer to Bent ReLU. So the proposed method in this paper will hopefully be able to employ any such progress, e.g. [30].

In all our case studies, the activation functions of the input and output layers are *identity maps*, whereas the activation functions of the hidden layers are set to be Bent ReLU with $\epsilon = 0.0001$, i.e.

$$a(x) = 0.5 \cdot x + \sqrt{0.25 \cdot x^2 + 0.0001} \tag{4}$$

and accordingly

$$a'(x) = 0.5 + \frac{0.25 \cdot x}{\sqrt{0.25 \cdot x^2 + 0.0001}} \quad . \tag{5}$$

3.1.2 Forward and Backward Propagation. Denote the input vector to the NN by $\mathbf{x} \in \mathbb{R}^n$. Let the output vector of the l-th layer be $\mathbf{x}^{(l)}$. Then $\mathbf{x}^{(0)} = \mathbf{x}$. We introduce the vector variable $\mathbf{z}^{(l)}$ to denote the input vector to the l-th layer for $1 \le l \le L$. Thus the *forward propagation equations* of NN can be defined as

$$\begin{cases} \mathbf{x}^{(0)} &= \mathbf{x} \\ \mathbf{z}^{(l)} &= \mathbf{W}^{(l)} \cdot \mathbf{x}^{(l-1)} + \mathbf{b}^{(l)} & \text{for } 1 \le l \le L \\ \mathbf{x}^{(l)} &= a^{(l)}(\mathbf{z}^{(l)}) & \text{for } 1 \le l \le L \\ y &= \mathbf{x}^{(L)} \end{cases} \tag{6}$$

where $\mathbf{x}$ is an n-dimensional column vector, $\mathbf{W}^{(l)}$ is a matrix of dimension $d^{(l)} \times d^{(l-1)}$, and $\mathbf{b}^{(l)}$ is a $d^{(l)}$-dimensional column vector; $a^{(L)}$ is the identity map, and $a^{(l)}$ for $1 \le l \le L-1$ is defined by (4) and taken as an element-wise function; y is the scalar output of the NN with input $\mathbf{x}$, and is identified as an abbreviation of the function $y(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$.

For the function $y(\mathbf{x})$ to be a barrier, by Theorem 2.3, it is crucial to analyze its Lie derivative $\mathcal{L}_{\mathbf{f}} y$, or its gradient ∇y. From (6) and by applying the chain-rule of derivative, we can deduce the *backward propagation equations* of the NN as follows:

$$\begin{cases} \frac{\partial y}{\partial \mathbf{z}^{(L)}} &= 1 \\ \frac{\partial y}{\partial \mathbf{z}^{(l-1)}} &= \left((\mathbf{W}^{(l)})^T \cdot \frac{\partial y}{\partial \mathbf{z}^{(l)}}\right) \odot a'(\mathbf{z}^{(l-1)}) \quad \text{for } 2 \le l \le L \\ \frac{\partial y}{\partial \mathbf{x}} &= (\mathbf{W}^{(1)})^T \cdot \frac{\partial y}{\partial \mathbf{z}^{(1)}} \end{cases} \tag{7}$$

where $\mathbf{z}^{(l)}$ is computed from (6), $\odot$ denotes the element-wise product of two vectors, i.e., the *Hadamard product*, and $a'(x)$ is defined as (5) and is taken as an element-wise function when applied to vectors.

For a given NN and an input $\mathbf{x}$, we can compute $y(\mathbf{x})$ and $\nabla y(\mathbf{x})$ according to (6) and (7), and thus $\mathcal{L}_{\mathbf{f}} y(\mathbf{x})$ according to (1). Therefore the two sets of propagation equations are cornerstone of our approach to learn and formally verify barrier certificates.

Example 3.1. **Prajna07** [36]. We use the following system as a running example to demonstrate our approach.

$$\mathbf{f} : \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_1 + \frac{1}{3}x_1^3 - x_2 \end{bmatrix},$$

- X_D: $\{\mathbf{x} \in \mathbb{R}^2 | -3.5 \le x_1 \le 2, -2 \le x_2 \le 1\}$;
- X_I: $\{\mathbf{x} \in \mathbb{R}^2 | (x_1 - 1.5)^2 + x_2^2 \le 0.25\}$;
- X_U: $\{\mathbf{x} \in \mathbb{R}^2 | (x_1 + 1)^2 + (x_2 + 1)^2 \le 0.16\}$.

An NN with two input neurons and one output neuron will be constructed as a barrier candidate. For the purpose of showing the propagation process, we temporarily assume that there is one hidden layer with two neurons, and assign values to weights and biases which are labeled on the arrows and neuron nodes in Fig. 2:

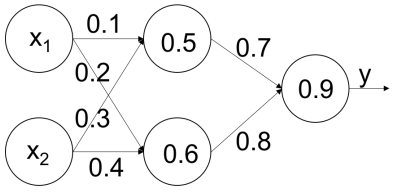

Figure 2: An NN constructed for Example 3.1

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.4 \end{bmatrix}, \mathbf{b}^{(1)} = \begin{bmatrix} 0.5 \\ 0.6 \end{bmatrix}, \mathbf{W}^{(2)} = \begin{bmatrix} 0.7 \\ 0.8 \end{bmatrix}^T, \mathbf{b}^{(2)} = \begin{bmatrix} 0.9 \end{bmatrix}$$

and $a^{(1)}$ is given by (4). Given input $\mathbf{x} = (1, 0)^T$, we can compute from (6) and (7) that $y(\mathbf{x}) = 1.9602$ and $\nabla y(\mathbf{x}) = (0.2300, 0.5299)^T$. Then it can be deduced that $\mathcal{L}_{\mathbf{f}} y(\mathbf{x}) = -0.3533$.

3.2 Training Data Generation

To learn a barrier certificate for $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, we sample from X_D, X_I and X_U separately to generate three sets of data points, which will play different roles in the loss function. Although both X_I and X_U are assumed to be subsets of X_D, treating them differently allows more flexible control of the size of the three generated data sets. In the sequel, we show how to sample data points from X_D and sampling of X_I and X_U follows in the same manner.

Mesh generation. Suppose $X_D \subseteq [\mathbf{l}, \mathbf{u}]_D$, where $\mathbf{l}, \mathbf{u} \in \mathbb{R}^n$ are lower and upper bounds of X_D, i.e., for all $\mathbf{x} \in X_D$, $\mathbf{x} \in [\mathbf{l}, \mathbf{u}]_D$ holds element-wisely. We then sample from each dimension of $[\mathbf{l}, \mathbf{u}]_D$ equidistantly with a fixed mesh size δ_i for $i \in [n]$, or equivalently, we can provide a vector of integers $\mathbf{n}^D_{\text{mesh}}$ specifying the number of points to be sampled from each dimension.

Mini-batch generation. For high-dimensional systems, the resulting mesh of $[\mathbf{l}, \mathbf{u}]_D$ has a huge size. As a common practice in machine learning, the data mesh will be partitioned into mini-batches, each with a small size. The advantages of using mini-batch training lie in allowing neural network training on a large data set, and, more specific to the barrier synthesis, it could facilitate *fine tuning*

of the shape of the barrier function, which will be demonstrated in Section 4. More concretely, we partition the mesh of $[\mathbf{l}, \mathbf{u}]_D$ into mini-batches by providing a vector of integers $\mathbf{n}^D_{\text{batch}}$, specifying the number of partitions in each dimension.

Filtering. Let $\prod \mathbf{x}$ denote the product of elements of $\mathbf{x}$. Then we have generated $\prod \mathbf{n}^D_{\text{mesh}}$ data points from X_D which are partitioned into $\prod \mathbf{n}^D_{\text{batch}}$ mini-batches. If X_D is identical to the super-rectangle $[\mathbf{l}, \mathbf{u}]_D$, then the training data generation from X_D has been completed. Otherwise, suppose that $X_D = \{\mathbf{x} \in \mathbb{R}^n | \phi(\mathbf{x})\}$, and then the generated mini-batches are sent to a filter to remove those data points violating ϕ.

Finally, a list of mini-batches bl_D comprising sampled data from X_D is obtained, which can be expressed as

$$bl_D := \text{filter}\Big(\text{batch}\big(\text{mesh}([\mathbf{l}, \mathbf{u}]_D, \mathbf{n}^D_{\text{mesh}}), \mathbf{n}^D_{\text{batch}}\big), \phi(X_D)\Big),$$

where mesh, batch, filter denote the operations for mesh generation, mini-batch generation and filtering. The lists of mini-batches bl_I and bl_U for X_I and X_U can be obtained similarly.

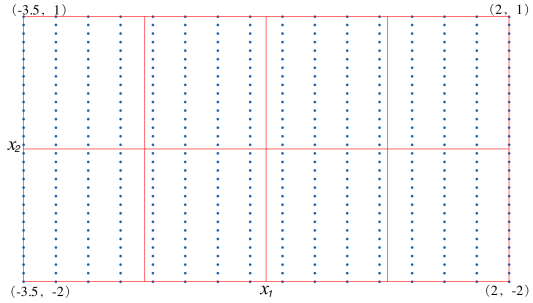

Figure 3: Illustration of batches of data for Example 3.1

Example 3.1 continued. Figure 3 shows the generated data points and batches with $\mathbf{n}^D_{\text{mesh}} = (16, 32)$ and $\mathbf{n}^D_{\text{batch}} = (4, 2)$ for X_D.

3.3 Loss Function

The core of our barrier certificate learning method is encoding of the three constraints in Theorem 2.3 into loss functions of NN. During the learning process, loss value being reduced to 0 indicates that the current learned weights and biases result a barrier certificate in the sense of Theorem 2.3. Given three finite training data sets $S_D \subseteq X_D$, $S_I \subseteq X_I$, and $S_U \subseteq X_U$, our loss function can be expressed as:

$$C(S_D, S_I, S_U) = \sum_{\mathbf{x} \in S_D} C_D(\mathbf{x}) + \sum_{\mathbf{x} \in S_I} C_I(\mathbf{x}) + \sum_{\mathbf{x} \in S_U} C_U(\mathbf{x}) \quad (8)$$

where C_I, C_U, and C_D are sub-loss functions from $\mathbb{R}^n$ to $\mathbb{R}$, related to the initial constraint, the unsafe constraint, and the Lie derivative constraint in Theorem 2.3, respectively. These three sub-loss functions will be formally defined below. Since the barrier function is to be represented by an NN, in the rest of this paper, we will replace the $B(\mathbf{x})$ in Theorem 2.3 by $y(\mathbf{x})$ with y defined in (6).

Encoding initial constraint. The intuition of defining C_I is to force $y(\mathbf{x})$ to have nonpositive values on $\mathbf{x} \in S_I$, by penalizing those $\mathbf{x} \in S_I$ with $y(\mathbf{x}) > 0$. The simplest way to achieve this is to let

$$C_I(\mathbf{x}) = \text{ReLU}(y(\mathbf{x}) + \tau_I) \qquad \text{for } \mathbf{x} \in S_I\,, \quad (9)$$

where $\tau_I \geq 0$ is a small non-negative constant. Roughly speaking, C_I is defined as the composition of ReLU and the output function y of the considered NN. It is obvious that

$$\forall \mathbf{x} \in S_I.\ (C_I(\mathbf{x}) = 0 \Rightarrow y(\mathbf{x}) \leq 0)\ . \quad (10)$$

The tolerance τ_I in (9) is to make the post-condition $y(\mathbf{x}) \leq 0$ in (10) holds in a neighbourhood of $\mathbf{x}$ for any $\mathbf{x}$ satisfying $C(\mathbf{x}) = 0$.

Encoding unsafe constraint. The sub-loss function C_U can be defined in the same manner as C_I:

$$C_U(\mathbf{x}) = \text{ReLU}(-y(\mathbf{x}) + \tau_U) \qquad \text{for } \mathbf{x} \in S_U\,, \quad (11)$$

where $\tau_U > 0$ is a small positive tolerance for guaranteeing the strict positivity of $y(\mathbf{x})$ on S_U. Likewise,

$$\forall \mathbf{x} \in S_U.\ (C_U(\mathbf{x}) = 0 \Rightarrow y(\mathbf{x}) > 0)\ . \quad (12)$$

Encoding Lie derivative constraint. The encoding of the Lie derivative constraint in Theorem 2.3 is slightly different from C_I and C_U, as such a constraint is only relevant to the points $\mathbf{x}$ on the 0-level set of $y(\mathbf{x})$. Since in practice it is hard to sample points satisfying $y(\mathbf{x}) = 0$ exactly, we consider a belt (or tube) region around the boundary $y(\mathbf{x}) = 0$. Thus C_D can be defined as

$$C_D(\mathbf{x}) = \begin{cases} \text{ReLU}\big(\mathcal{L}_{\mathbf{f}} y(\mathbf{x}) + \tau_D\big) & \text{if } |y(\mathbf{x})| \leq \tau_B \\ 0 & \text{otherwise} \end{cases} \quad (13)$$

where $\tau_D > 0$ is a small positive tolerance for guaranteeing the strict negativity of $\mathcal{L}_{\mathbf{f}} y(\mathbf{x})$ for $\mathbf{x}$ satisfying $y(\mathbf{x}) = 0$; $\tau_B > 0$ is a positive constant specifying the width of the belt region around $y(\mathbf{x})$. It is obvious that

$$\forall \mathbf{x} \in S_D.\ \big(C_D(\mathbf{x}) = 0 \wedge y(\mathbf{x}) = 0 \Rightarrow \mathcal{L}_{\mathbf{f}} y(\mathbf{x}) < 0\big)\ . \quad (14)$$

Example 3.1 continued. For the NN shown in Fig. 2 with input $\mathbf{x} = (1, 0)$, if $\tau_I = \tau_U = \tau_D = 0$ and $\tau_B = 0.05$, then we have $C_I(\mathbf{x}) = 1.9602$, $C_D(\mathbf{x}) = 0$, and $C_U(\mathbf{x})$ is not applicable.

The following proposition shows that the loss function C defined above well characterizes the conditions of Theorem 2.3.

Proposition 3.2. *Given a system $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, if the output function $y(\mathbf{x})$ of an NN constructed for Γ satisfies $C(S_D, S_I, S_U) = 0$ for any* finite *sets $S_D \subseteq X_D, S_I \subseteq X_I, S_U \subseteq X_U$, where C is defined by (8), (9), (11) and (13), then $y(\mathbf{x})$ is a barrier certificate of Γ.*

Proof. By the proof method of contradiction, the result follows from (10), (12), (14) and the three conditions from Theorem 2.3. □

By Proposition 3.2, given a system Γ, if the training data set in Subsection 3.2 is appropriately generated and an NN is trained to obtain 0 loss value on the training set with the loss function C defined by (8), then it is likely that such an NN satisfies the condition of Proposition 3.2 and thus is quantified to be a barrier certificate of Γ. This indicates that the barrier certificate generation problem can be reduced to the well-studied NN training problem. Furthermore, by Proposition 3.3, the loss function C has the property that once the loss is decreased to 0 it will stabilize, which suggests a good terminating criterion for the training process.

Proposition 3.3. *Given an NN with the loss function C defined in (8), when C evaluates to 0 on the full set of training data, the NN will not be updated in subsequent training by gradient descent methods.*

Proof. Suppose all the weights and biases of the NN can be denoted by $\mathbf{W}$ and $\mathbf{b}$, and accordingly the loss function C in (8) on the full data set can be expressed as $C(\mathbf{W}, \mathbf{b})$. From (9), (11) and (13) and by the chain rule of derivative and the derivative of ReLU, it is easy to show that $\frac{\partial C}{\partial \mathbf{W}}$ and $\frac{\partial C}{\partial \mathbf{b}}$ are both zero vectors. As a result, $\mathbf{W}$ and $\mathbf{b}$ will not be updated and the training stabilizes. □

By Proposition 3.2, in NN-learning based barrier certificate synthesis, the chance of successfully synthesizing a barrier certificate depends on the quality of the generated training set. In the rest of the paper, we will present the detailed synthesis algorithm, and discuss the issues regarding increasing chances of successful synthesis and formally guaranteeing the correctness of the generated barrier certificates.

3.4 The Learning Algorithm

3.4.1 The Main Algorithm. Our main learning algorithm is illustrated in Algorithm 1.

Algorithm 1 Barrier Certificate Learning

Input: $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, L, $\mathbf{d}$, $\mathbf{n}^D_{\text{mesh}}$, $\mathbf{n}^I_{\text{mesh}}$, $\mathbf{n}^U_{\text{mesh}}$, $\mathbf{n}^D_{\text{batch}}$, $\mathbf{n}^I_{\text{batch}}$, $\mathbf{n}^U_{\text{batch}}$, n_{restart}, n_{epoch}, τ_I, τ_U, τ_D, τ_B, l_r, u_∇;
Output: $\mathbf{W}$, $\mathbf{b}$ of the learned NN;

```
1:  W, b = nn_construct(Γ, L, d);
2:  bl_I, bl_U, bl_D = data_gen(Γ, n^{D,I,U}_mesh, n^{D,I,U}_batch);
3:  for i = 1 to n_restart do
4:      initialize(W, b);
5:      for j = 1 to n_epoch do
6:          C_epoch = 0;
7:          shuffle(bl_I, bl_U, bl_D);
8:          for S_I, S_U, S_D in bl_I, bl_U, bl_D do
9:              C_batch = loss(S_I, S_U, S_D, τ_I, τ_U, τ_D, τ_B);
10:             C_epoch += C_batch;
11:             update_gradient_descent(W, b, l_r);
12:             gradient_control(u_∇);
13:         end for
14:         if decide_success(C_epoch, u_∇) then
15:             return W, b;
16:         end if
17:     end for
18: end for
```

The details of Algorithm 1 and its subroutines are given as follows:

- **Inputs of the algorithm**: Γ is the considered system to be verified; L specifies the number of layers of the NN to be constructed and $\mathbf{d}$ specifies the number of neurons in $L-1$ hidden layers (nn_construct in Line 1); $\mathbf{n}^{D,I,U}_{\text{mesh}}$ and $\mathbf{n}^{D,I,U}_{\text{batch}}$ are shorthand for 6 integer vectors specifying the number of sampling points and the number of batches for each dimension when generating training data (data_gen in Line 2); since we adopt mini-batch training, one traversal of all the mini-batches is called an *epoch*, and n_{epoch} specifies how many passes we need to traverse the full data set (Line 5 to 17); since the performance of neural network training depends on the weights and biases initialization (initialize in Line 4 adopts standard Gaussian distribution), we specify how many times we want to re-initialize the NN parameters by n_{restart} (Line 3 to 18); $\tau_I, \tau_U, \tau_D, \tau_B$ are the four tolerances used in the computation of loss values C_{batch} according to (8) for each mini-batch 3-tuple (S_I, S_U, S_D) (loss in Line 9); $l_r \in \mathbb{R}$ specifies a constant *learning rate* for the *gradient descent* optimization method used to update the weights and biases (update_gradient_descent in Line 11);
- **Output of the algorithm**: if the algorithm succeeds within the specified number of restarts, the learned weights $\mathbf{W}$ and biases $\mathbf{b}$ will be returned (Line 15);
- **Data shuffling**: as a common practice in NN training using *stochastic gradient descent* (SGD) method, the training data set is shuffled at the beginning of each epoch; here to keep the *locality* of training data whose advantage will be demonstrated in Section 4, we adopt limited shuffling, that is, we shuffle the list of mini-batches in bl_I, bl_U, bl_D instead of the full data set (shuffle in Line 7);
- **Gradient control**: the input $u_\nabla \in \mathbb{R}$ is used as an upper bound in the gradient_control sub-routine (Line 12) for controlling the gradient norm of the learned NN; and the return condition checks not only whether $C_{\text{epoch}} = 0$ but also the norm of ∇y w.r.t. u_∇ (decide_success in Line 14), the reason for which is to avoid generating a *false* barrier certificate and we will investigate this in depth in the following subsection.

3.4.2 Gradient Control. The importance of gradient control in Algorithm 1 can be illustrated by Fig. 4. Intuitively, to learn a barrier

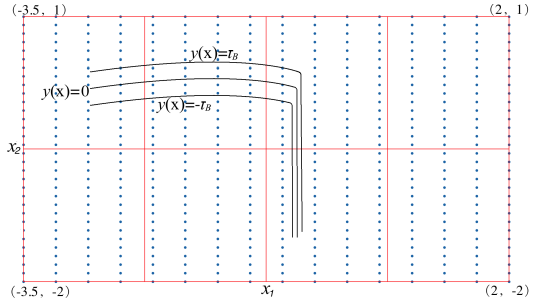

Figure 4: Poor sampling may result in false barriers

certificate, by Proposition 3.2, we need to reduce $C_D(\mathbf{x})$ to 0 for all $\mathbf{x}$ in the belt $|y(\mathbf{x})| \le \tau_B$. Since our approach is data-driven, it demands that sufficiently many points are sampled in the belt $|y(\mathbf{x})| \le \tau_B$, and these points should be scattered evenly in the belt. However, if the belt is so narrow that it is entirely or partly located in the gaps of the sampled data points as shown in Fig. 4, then the Lie derivative $\mathcal{L}_\mathbf{f} y$ on the boundary $y(\mathbf{x}) = 0$ will (partly) not be constrained by the sampled data at all, and thus a false barrier may be learned. The vertical part of $y(\mathbf{x}) = 0$ in Fig. 4 may suffer from such risks.

The simplest way to avoid generating a false barrier is to control the norm of ∇y within a specified upper bound u_∇ so that the belt $|y(\mathbf{x})| \le \tau_B$ will be wide enough compared to the mesh size of data

sampling, as we do in Algorithm 1. The idea is as follows. Consider an arbitrary $\mathbf{x}_0 \in X_D$ s.t. $y(\mathbf{x}_0) = 0$.[2] Then the Taylor expansion of $y(\mathbf{x})$ at $\mathbf{x}_0$ gives that $y(\mathbf{x}) = \nabla y(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0) + o(\|\mathbf{x} - \mathbf{x}_0\|_2)$, where $\|\cdot\|_2$ denotes the Euclidean norm. By the triangle inequality and the Cauchy-Schwarz inequality we have

$$\begin{array}{rcl} |y(\mathbf{x})| & \leq & |\nabla y(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0)| + o(\|\mathbf{x} - \mathbf{x}_0\|_2) \\ & \leq & \|\nabla y(\mathbf{x}_0)\|_2 \cdot \|\mathbf{x} - \mathbf{x}_0\|_2 + o(\|\mathbf{x} - \mathbf{x}_0\|_2) \\ & \leq & \sqrt{n} \cdot \|\nabla y(\mathbf{x}_0)\|_2 \cdot \|\mathbf{x} - \mathbf{x}_0\|_\infty + o(\|\mathbf{x} - \mathbf{x}_0\|_2) \end{array} \quad (15)$$

The last inequality is by $\|\cdot\|_2 \leq \sqrt{n}\,\|\cdot\|_\infty$ where $\|\cdot\|_\infty$ denotes the ℓ_∞ norm, i.e. $\|\mathbf{x}\|_\infty = \max_{i\in[n]} |x_i|$. Assume a mesh size δ when generating training data. Thus $\mathbf{x}_0$ is located in a hyper-rectangle with side length δ. Now consider a vertex $\mathbf{x}_1$ of the rectangle at which $\mathbf{x}_0$ is located, then $\|\mathbf{x}_1 - \mathbf{x}_0\|_\infty \leq \delta$. Thus from (15) we can have a good upper bound estimate of $|y(\mathbf{x}_1)|$ by $\sqrt{n}\delta\,\|\nabla y(\mathbf{x}_0)\|_2$. Based on this estimate, if the upper bound u_∇ of $\|\nabla y\|_2$ is chosen such that

$$u_\nabla \leq \frac{\tau_B}{\sqrt{n}\delta} \quad , \quad (16)$$

then $|y(\mathbf{x}_1)|$ will have an upper bound estimate of τ_B, which implies that all the sampled points in the neighbourhood of $\mathbf{x}_0$ will have high possibility of falling in the belt $|y(\mathbf{x})| \leq \tau_B$. In practice, we will relax the bound given by (16) to some extent.

Given u_∇ in accordance with (16), the gradient_control subroutine in Algorithm 1 detects whether the norm of ∇y exceeds u_∇, and if so, will bring it down below u_∇. The mechanism of gradient_control is to scale the weights and biases of the currently learned NN. The basic idea is: given a system $\Gamma = (\mathbf{f}, X_D, X_I, X_U)$, if $B(\mathbf{x})$ is a barrier certificate of Γ as per Theorem 2.3, then it is obvious that $c \cdot B$ is also a barrier certificate of Γ for any constant $c > 0$. This indicates that scaling an NN with small positive c will reduce the norm of the gradient of the output function without affecting barrier certificate learning. In our implementation of Algorithm 1, scaling is based on the following lemma:

Lemma 3.4. *Consider an NN with the same structure as the one in Subsection 3.1 except that all the activation functions in the hidden layers are replaced by ReLU. Given a positive constant $\alpha > 0$, if we reset the weight matrix $\mathbf{W}^{(l)}$ to $\alpha \cdot \mathbf{W}^{(l)}$ and reset the bias vector $\mathbf{b}^{(l)}$ to $\alpha^l \cdot \mathbf{b}^{(l)}$, for all $1 \leq l \leq L$. Then the new output function will be $\alpha^L \cdot y(\mathbf{x})$ where $y(\mathbf{x})$ is the original output before reset.*

The proof of Lemma 3.4 is straightforward and thus omitted here. Lemma 3.4 indicates a specific scaling approach for NN with ReLU activations, by scaling the weights and biases of each layer using different factors. Since Bent ReLU has a similar shape to ReLU, we adopt the same scaling approach in Algorithm 1, by choosing the scale constant $\alpha = 0.5$. This means, for an NN with one hidden layer, the scaled $y(\mathbf{x})$ would be *roughly* $\frac{1}{4}$ of the original one.

Example 3.1 continued. For Example 3.1, by constructing an NN with one hidden layer consisting of 5 neurons, a barrier certificate is successfully generated. In Fig. 5, the green and red circles represent the initial and unsafe regions, the arrows represent the vector field, and the three solid curves represent the $|y| \leq \tau_B$ belt of the learned barrier, i.e., the τ_B, 0, and $-\tau_B$-level sets of y. The returned weights and biases are:

$$\mathbf{W}^{(1)} = \begin{bmatrix} 0.1729 & -0.3968 & -0.6386 & -0.3343 & -0.7811 \\ -0.7008 & -0.1161 & 0.2372 & -0.4997 & 0.2423 \end{bmatrix}^T,$$

$$\mathbf{b}^{(1)} = \begin{bmatrix} -0.1114 & -0.7368 & -1.1864 & -0.1170 & -0.9058 \end{bmatrix}^T,$$

$$\mathbf{W}^{(2)} = \begin{bmatrix} 0.2850 & -0.9105 & 0.5477 & 0.3664 & -0.6661 \end{bmatrix},$$

$$\mathbf{b}^{(2)} = \begin{bmatrix} -0.1962 \end{bmatrix}$$

For this example, $\tau_B = 0.05$, and the mesh sizes of X_D are $\delta_{x_1} = 0.0215$ and $\delta_{x_2} = 0.0117$. Then $\frac{\tau_B}{\sqrt{2}\delta_{x_1}} = 1.645$ and $\frac{\tau_B}{\sqrt{2}\delta_{x_2}} = 3.017$ by (16). In Algorithm 1, we relaxed the upper bound of ∇y to $u_\nabla = 6$.

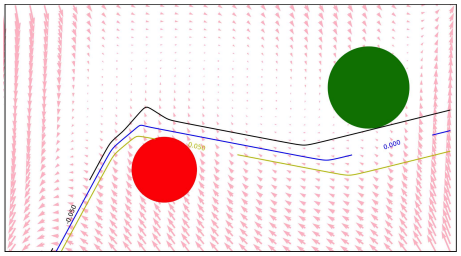

Figure 5: Barrier certificate generated for Example 3.1

4 FORMAL VERIFICATION

The *gradient control* technique in Algorithm 1 significantly reduces the risk of generating a false barrier certificate. However, it does not provide a formal guarantee. In this section, we present an approach to formally check whether the learned function, represented by the neural network, can indeed serve as a barrier certificate. That is, given a candidate barrier function $B(\mathbf{x})$ in the form of an NN output function $y(\mathbf{x})$, we verify the three conditions of Theorem 2.3 hold, or equivalently, the negation of the conditions are unsatisfiable:

$$\begin{cases} \exists \mathbf{x}.\, \mathbf{x} \in X_I \wedge y(\mathbf{x}) > 0 \\ \exists \mathbf{x}.\, \mathbf{x} \in X_U \wedge y(\mathbf{x}) \leq 0 \\ \exists \mathbf{x}.\, \mathbf{x} \in X_D \wedge y(\mathbf{x}) = 0 \wedge \mathcal{L}_{\mathbf{f}} y(\mathbf{x}) \geq 0 \end{cases} \quad (17)$$

Essentially, (17) can be encoded into SMT constraints and is a range estimation problem. Note the computation of $y(\mathbf{x})$ and $\mathcal{L}_{\mathbf{f}} y(\mathbf{x})$ in (17) by the forward and backward equations (6) and (7) involves the computation of nonlinear functions $a(x)$ and $a'(x)$, which are the hard-core of formal verification. Our strategies for tackling the problem are as follows.

Symbolic Approach. As claimed in Subsection 3.1.1, the Bent ReLU activation function enables the application of symbolic computation in the verification of NN. To see this, we rewrite (2) and (3) into the following polynomial forms:

$$\begin{cases} (a - 0.5 \cdot x)^2 = 0.25 \cdot x^2 + \epsilon \wedge a > 0 \\ (a' - 0.5) \cdot (a - 0.5 \cdot x) = 0.25 \cdot x \end{cases} \quad (18)$$

where a and a' are shorthand of $a(x)$ and $a'(x)$ respectively. It is not difficult to check that (18) is equivalent to the conjunction of (2) and (3). If X_D, X_I, X_U and $\mathbf{f}$ are all described by polynomial expressions, then (6), (7), (17) and (18) together form an existentially

[2] If $y(\mathbf{x})$ is a barrier function, then it has opposite signs on X_I and X_U so that $\mathbf{x}_0$ usually exists.

quantified polynomial formula, the truth value of which can be obtained by *quantifier elimination* (QE [5]). However, due to its high computational complexity [7], QE is applied restrictedly in our approach, as shown in the following paragraph.

Piecewise Linear Approximation. It was also claimed in Subsection 3.1.1 that the shape of Bent ReLU activation function is quite similar to (piece-wise linear) ReLU, which suggests that we could compute a precise piece-wise linear (over-)approximation of Bent ReLU, to simplify the computation of a and a'. For instance, $a(x)$ in (4) could be easily approximated by 4 pieces as $\bar{a}(x)$:

$$\bar{a}(x) \in \begin{cases} (0,\ 0.001926) & \text{if } x \in (-\infty,\ -0.05] \\ (0.001925,\ 0.01] & \text{if } x \in (-0.05,\ 0] \\ (x + 0.001925,\ x + 0.01) & \text{if } x \in (0,\ 0.05) \\ (x,\ x + 0.001926) & \text{if } x \in [0.05,\ \infty) \end{cases} \tag{19}$$

The approximation error between $a(x)$ and $\bar{a}(x)$ is at most 0.01. If we increase the number of pieces, higher approximation precisions will be achieved. In our verification practice, we use 2, 4 or 6 pieces to approximate $a(x)$ and 6 pieces to approximate $a'(x)$. The details are omitted here due to space limitation. Note that as in (19), linearized formulas usually involve floating-point numbers due to Taylor expansion. In our work, all numerically computed approximations are symbolically verified using (18) and QE. Our linearizations, together with (6), (7) and (17), form a constraint in which most expressions are linear and nonlinearity may only exist in the expressions defining X_D, X_I, X_U and $\mathbf{f}$. In our work, the finally resulting problem is sent to isat3, an interval constraint propagation (ICP) based nonlinear SMT solver for verification.[3]

Example 3.1 continued. The barrier function shown in Fig. 5 is successfully verified using isat3 with 2-pieces approximation of a and 6-pieces approximation of a'.

Combine Pre-training and Fine-tuning. In our encoding of loss functions in Section 3.3, four tolerances $\tau_I, \tau_U, \tau_D, \tau_B$ are used so that points not sampled will also have zero loss, but generally there is no good principle on how to choose the tolerances. If $\tau_I, \tau_U, \tau_D, \tau_B$ are not large enough, even a true barrier certificate may fail the verification due to error propagation of interval computation and the approximation error of linearization. Our strategy for tackling this problem is to perform pre-training with very small tolerances (or even zero tolerances except for τ_B) to get a first solution. Then we iteratively increase the tolerances and perform fine-tuning of the NN initialized by the pre-trained solutions. In practice, fine-tuning works very well in that it costs much less time than pre-training, and the increased tolerances enable us to succeed in formal verification. The reason for the effectiveness of fine-tuning may be that our mini-batch training keeps the locality of data so the shape of the barrier can be tuned in a fine granularity. The experiment details are reported in the next section.

Example 3.1 continued. In Fig. 6, the left figure shows the pre-trained $|y| \leq 0.05$ belt with tolerances $\tau_I = \tau_U = \tau_D = 0, \tau_B = 0.05$, which is learned from a randomized initialization, while the right figure shows the fine-tuned $|y| \leq 0.05$ belt with the safety tolerance τ_U increased to 0.05. It is evident that verification of the fine-tuned one is much easier.

[3]https://projects.informatik.uni-freiburg.de/projects/isat3/

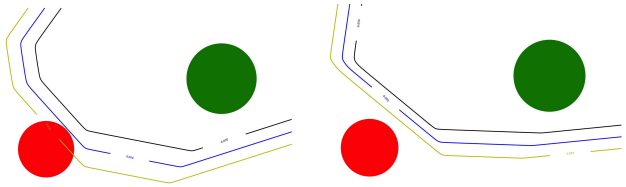

Figure 6: Illustration of pre-training and fine-tuning

5 CASE STUDIES

We have implemented a prototype tool named nnbarrier based on Algorithm 1 using the PyTorch[4] platform. We apply nnbarrier to verify a number of cases from the literature. All experiments are performed on a laptop workstation running Ubuntu 18.04 with Intel i7-8550u CPU and 32GB memory. The tool package together with a short instruction, as well as all the case studies, are publicly available.[5]

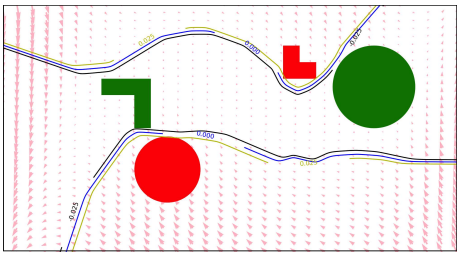

Figure 7: The learned barrier for Example 5.1

Example 5.1. **Prajna07-modified**. This example is to show the flexibility of our method in dealing with multiple or irregular initial and unsafe regions.

$$\mathbf{f}: \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} x_2 \\ -x_1 + \frac{1}{3}x_1^3 - x_2 \end{bmatrix},$$

- X_D: $\{\mathbf{x} \in \mathbb{R}^2 : -3.5 \leq x_1 \leq 2, -2 \leq x_2 \leq 1\}$;
- X_I: $\{\mathbf{x} \in \mathbb{R}^2 : (x_1 - 1.5)^2 + x_2^2 \leq 0.25 \vee (x \geq -1.8 \wedge x \leq -1.2 \wedge y \geq -0.1 \wedge y \leq 0.1) \vee (x \geq -1.4 \wedge x \leq -1.2 \wedge y \geq -0.5 \wedge y \leq 0.1)\}$;
- X_U: $\{\mathbf{x} \in \mathbb{R}^2 : (x_1+1)^2+(x_2+1)^2 \leq 0.16 \vee (x \geq 0.4 \wedge x \leq 0.6 \wedge y \geq 0.1 \wedge y \leq 0.5) \vee (x \geq 0.4 \wedge x \leq 0.8 \wedge y \geq 0.1 \wedge y \leq 0.3)\}$.

Using an NN with one hidden layer consisting of 20 neurons, a barrier certificate was generated as shown in Fig. 7, in which initial and unsafe sets are colored green and red respectively, and the $|y| \leq 0.025$ belt of the learned barrier is shown. Note that it would be rather complicated to encode this example into SOS-constraints using SOS-based methods.

Example 5.2. **Darboux** [48]. This example is to show the strength of our approach by encoding the strict barrier certificate condition.

$$\mathbf{f}: \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} x_2 + 2x_1x_2 \\ -x_1 + 2x_1^2 - x_2^2 \end{bmatrix},$$

[4]https://pytorch.org/

[5]https://github.com/zhaohj2017/HSCC20-Repeatability

Table 1: Inputs and time costs for barrier certificate training and verification

	L	$\mathbf{d}$	$\mathbf{n}^D_{\text{mesh}}$	$\mathbf{n}^D_{\text{batch}}$	n_{r}	n_{e}	τ_I	τ_U	τ_D	τ_B	l_r	u_∇	$T_{\text{nnbarrier}}$	τ^f_I	τ^f_U	τ^f_D	τ^f_B	T_{isat3}
Ex 5.1	2	20	$(2^8, 2^8)$	$(2^6, 2^6)$	5	500	0	0	0.001	0.05	0.01	6	1731.03s	0.01	0.01	0.03	0.025	635.31s
Ex 5.2	2	10	$(2^8, 2^8)$	$(2^6, 2^6)$	5	100	0	0	0.005	0.05	0.1	6	341.45s	0.025	0.025	0.005	0.05	20.84s
Ex 5.3	2	10	$(2^8, 2^8)$	$(2^6, 2^6)$	5	500	0	0	0.001	0.05	0.1	6	637.48s	0.02	0.02	0.10	0.05	11.30s
Ex 5.4	2	10	$(2^7, 2^7, 2^7)$	$(2^4, 2^4, 2^4)$	5	100	0	0	0	0.05	0.01	6	3165.28s	0.05	0.07	0.15	0.10	1003.37s

- X_D: $\{\mathbf{x} \in \mathbb{R}^2 : -2 \le x_1, x_2 \le 2\}$;
- X_I: $\{\mathbf{x} \in \mathbb{R}^2 : 0 \le x_1 \le 1, 1 \le x_2 \le 2\}$;
- X_U: $\{\mathbf{x} \in \mathbb{R}^2 : x_1 + x_2^2 \le 0\}$.

It was reported [48] that LMI-based methods failed to verify this problem using polynomial template of degree 6, and we failed to verify this problem using the exponential barrier certificate condition [20] with polynomial template of degree 8.[6] However, by constructing an NN with one hidden layer consisting of 10 neurons, we successfully generate a barrier certificate.

Example 5.3. **Elementary** [28]. This example is to show the ability of our methods in dealing with non-polynomial systems.

$$\mathbf{f}: \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \end{bmatrix} = \begin{bmatrix} e^{-x_1} + x_2 - 1 \\ -\sin^2 x_1 \end{bmatrix},$$

- X_D: $\{\mathbf{x} \in \mathbb{R}^2 : -2 \le x_1, x_2 \le 2\}$;
- X_I: $\{\mathbf{x} \in \mathbb{R}^2 : (x_1 + 0.5)^2 + (x_2 - 0.5)^2 \le 0.16\}$;
- X_U: $\{\mathbf{x} \in \mathbb{R}^2 : (x_1 - 0.7)^2 + (x_2 + 0.7)^2 \le 0.09\}$.

In [28], the elementary ODE was transformed into an equivalent polynomial ODE with higher dimension. In contrast, our method can deal with it directly, and a barrier certificate was successfully generated by an NN with one hidden layer consisting of 10 neurons.

Example 5.4. **Obstacle avoidance** [1]. This example is to show the application of our method in control. The problem is to control a 2-dimensional airplane to avoid an obstacle by controlling its angular velocity.

$$\mathbf{f}: \begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v \sin\psi \\ v \cos\psi \\ u \end{bmatrix},$$

where ψ is the clockwise angle from the positive x_2-axis, v is the magnitude of velocity which is assumed to be 1, and the control law u is designed as:

$$u = -\sin\psi + 3 \cdot \frac{x_1 \cdot \sin\psi + x_2 \cdot \cos\psi}{0.5 + x_1^2 + x_2^2}$$

Intuitively, the plane is controlled to fly in the direction of positive x_2-axis, and meanwhile to avoid an obstacle centered at $(0, 0)$ by turning left.

- X_D: $\{\mathbf{x} \in \mathbb{R}^3 : -2 \le x_1, x_2 \le 2, -\pi/2 < \psi < \pi/2\}$;
- X_I: $\{\mathbf{x} \in \mathbb{R}^3 : -0.1 \le x_1 \le 0.1, -2 \le x_2 \le -1.8, -\pi/6 < \psi < \pi/6\}$;
- X_U: $\{\mathbf{x} \in \mathbb{R}^3 : x_1^2 + x_2^2 \le 0.04\}$.

Using an NN with one hidden layer consisting of 10 neurons, we successfully verified the obstacle avoidance property by generating a barrier certificate. In Fig. 8, the green cuboid, red cylinder, and yellow surface represent the initial set, unsafe set, and the 0-level set of the learned barrier, respectively; some simulated trajectories from the initial set are also shown in Fig. 8 as blue curves.

[6] Experiment was done in Matlab using Yalmip [29] and the Mosek SDP solver (https://www.mosek.com/), with $\lambda = 0, -0.5, -1$.

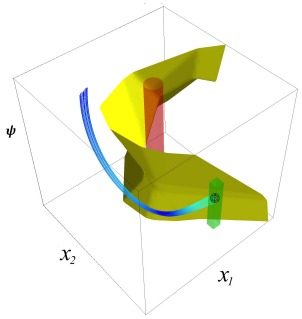

Figure 8: The learned barrier for Example 5.4

Statistics of our case studies are summarized in Table 1: n_{r} and n_{e} denote the number of restarts and epochs; $\mathbf{n}^{I,U}_{\text{mesh}}$ and $\mathbf{n}^{I,U}_{\text{batch}}$ are omitted; the tolerances $\tau_I, \tau_U, \tau_D, \tau_B$ are used for pre-training, while their counterparts with superscripts f are the finally fine-tuned ones; $T_{\text{nnbarrier}}$ measures the averaged time cost (in seconds) of pre-training for each case over 5 runs; T_{isat3} measures the time cost of verification in isat3 of the finally fine-tuned barrier certificates.

6 CONCLUSIONS

We have presented a data-driven approach for synthesis of barrier certificates for safety verification of general continuous dynamical systems. We proposed NN-type barrier certificates and introduced a new class of activation functions. We also gave methods to generate train sets, formulate the loss functions, together with regularization via gradient controls. Furthermore, we presented practical methods to formally verify the generated barrier certificates. The techniques can be adapted to hybrid systems, at least in principle. We believe the current work has made the first step towards exploiting NN in traditional verification methods for hybrid systems, and has greater potential to shed light on other applications of the same kind.

In the future, we shall explore the scalability to higher dimensions via *deeper* NNs and GPU training. Moreover, hyper-parameter tuning is challenging which may need more experiments. It is also interesting to explore the case when the (initial or unsafe) domains are unbounded. More generally, we believe verification would be a new, exciting application domain of NNs which merits further investigations.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] Andrew J. Barry, Anirudha Majumdar, and Russ Tedrake. 2012. Safety verification of reactive controllers for UAV flight in cluttered environments using barrier certificates. In *2012 IEEE International Conference on Robotics and Automation, ICRA 2012*. nstitute of Electrical and Electronics Engineers Inc., 484–490.

[2] Marc Brockschmidt, Yuxin Chen, Pushmeet Kohli, Siddharth Krishna, and Daniel Tarlow. 2017. Learning Shape Analysis. In *Static Analysis*. Springer International Publishing, 66–87.

[3] Mingshuai Chen, Jian Wang, Jie An, Bohua Zhan, Deepak Kapur, and Naijun Zhan. 2019. NIL: Learning Nonlinear Interpolants. In *Automated Deduction – CADE 27*. Springer International Publishing, 178–196.

[4] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. 2012. Taylor Model Flowpipe Construction for Non-linear Hybrid Systems. In *RTSS 2012*. IEEE Computer Society, Los Alamitos, CA, USA, 183–192.

[5] George E. Collins. 1975. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages*, H. Brakhage (Ed.). LNCS, Vol. 33. Springer Berlin Heidelberg, 134–183.

[6] Liyun Dai, Ting Gan, Bican Xia, and Naijun Zhan. 2017. Barrier Certificates Revisited. *Journal of Symbolic Computation* 80 (2017), 62–86.

[7] James H. Davenport and Joos Heintz. 1988. Real quantifier elimination is doubly exponential. *J. Symb. Comput.* 5, 1-2 (Feb. 1988), 29–35.

[8] A. Dolzmann, A. Seidl, and T. Sturm. 2006. *Redlog User Manual* (edition 3.1, for redlog version 3.06 (reduce 3.8) ed.). http://redlog.dolzmann.de/downloads/.

[9] Tommaso Dreossi, Daniel J. Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A. Seshia. 2019. VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems. In *Computer Aided Verification*. Springer International Publishing, 432–442.

[10] Souradeep Dutta, Xin Chen, and Sriram Sankaranarayanan. 2019. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC*. 157–168.

[11] Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. 2018. Output Range Analysis for Deep Feedforward Neural Networks. In *NASA Formal Methods*. Springer International Publishing, 121–138.

[12] P. Ezudheen, Daniel Neider, Deepak D'Souza, Pranav Garg, and P. Madhusudan. 2018. Horn-ICE learning for synthesizing invariants and contracts. *PACMPL* 2, OOPSLA (2018), 131:1–131:25. https://doi.org/10.1145/3276501

[13] Goran Frehse. 2008. PHAVer: algorithmic verification of hybrid systems past HyTech. *Int. J. Softw. Tools Technol. Transf.* 10, 3 (May 2008), 263–279.

[14] Goran Frehse, Colas Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *CAV 2011*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). LNCS, Vol. 6806. Springer Berlin Heidelberg, 379–395.

[15] Ting Gan, Mingshuai Chen, Liyun Dai, Bican Xia, and Naijun Zhan. 2015. Decidability of the Reachability for a Family of Linear Vector Fields. In *Automated Technology for Verification and Analysis*. Springer International Publishing, 482–499.

[16] Ting Gan, Mingshuai Chen, Y. Li, Bican Xia, and Naijun Zhan. 2018. Reachability Analysis for Solvable Dynamical Systems. *IEEE Trans. Automat. Control* 63, 7 (2018), 2003–2018.

[17] C. A. R. Hoare. 1969. An axiomatic basis for computer programming. *Commun. ACM* 12, 10 (Oct. 1969), 576–580.

[18] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. 2019. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019*. 169–178.

[19] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*. Springer, 97–117.

[20] Hui Kong, Fei He, Xiaoyu Song, William NN Hung, and Ming Gu. 2013. Exponential-condition-based barrier certificate generation for safety verification of hybrid systems. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV)*. Springer, 242–257.

[21] Hui Kong, Xiaoyu Song, Dong Han, Ming Gu, and Jiaguang Sun. 2014. A new barrier certificate for safety verification of hybrid systems. *Comput. J.* 57, 7 (2014), 1033–1045.

[22] Gerardo Lafferriere, George J. Pappas, and Sergio Yovine. 1999. A New Class of Decidable Hybrid Systems. In *HSCC 1999 (LNCS)*, Frits W. Vaandrager and Jan H. Schuppen (Eds.), Vol. 1569. Springer Berlin Heidelberg, 137–151.

[23] Moshe Leshno, Vladimir Ya. Lin, Allan Pinkus, and Shimon Schocken. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks* 6, 6 (1993), 861 – 867.

[24] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. 2019. Analyzing Deep Neural Networks with Symbolic Propagation: Towards Higher Precision and Faster Verification. In *Static Analysis*. Springer International Publishing, 296–319.

[25] Yi Li, Xuechao Sun, Yong Li, Andrea Turrini, and Lijun Zhang. 2019. Synthesizing Nested Ranking Functions for Loop Programs via SVM. In *Formal Methods and Software Engineering*. Springer International Publishing, 438–454.

[26] Jiang Liu, Jidong Lv, Zhao Quan, Naijun Zhan, Hengjun Zhao, Chaochen Zhou, and Liang Zou. 2010. A Calculus for Hybrid CSP. In *APLAS 2010*, Kazunori Ueda (Ed.). LNCS, Vol. 6461. Springer Berlin Heidelberg, 1–15.

[27] Jiang Liu, Naijun Zhan, and Hengjun Zhao. 2011. Computing semi-algebraic invariants for polynomial dynamical systems. In *EMSOFT 2011*. ACM, New York, NY, USA, 97–106.

[28] Jiang Liu, Naijun Zhan, Hengjun Zhao, and Liang Zou. 2015. Abstraction of Elementary Hybrid Systems by Variable Transformation. In *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*. Springer, 360–377.

[29] Johan Löfberg. 2004. YALMIP : A Toolbox for Modeling and Optimization in MATLAB. In *Proc. of the CACSD Conference*. Taipei, Taiwan. http://users.isy.liu.se/johanl/yalmip/.

[30] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. 2017. The Expressive Power of Neural Networks: A View from the Width. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. 6231–6239.

[31] Zohar Manna and Henny B. Sipma. 1998. Deductive verification of hybrid systems using STeP. In *HSCC 1998*, ThomasA. Henzinger and Shankar Sastry (Eds.). LNCS, Vol. 1386. Springer Berlin Heidelberg, 305–318.

[32] Ian Mitchell and Claire J. Tomlin. 2000. Level Set Methods for Computation in Hybrid Systems. In *HSCC 2000*, Nancy Lynch and Bruce H. Krogh (Eds.). LNCS, Vol. 1790. Springer Berlin Heidelberg, 310–323.

[33] André Platzer. 2010. Differential-algebraic Dynamic Logic for Differential-algebraic Programs. *J. Log. and Comput.* 20, 1 (Feb. 2010), 309–352.

[34] André Platzer and Edmund M. Clarke. 2008. Computing Differential Invariants of Hybrid Systems as Fixedpoints. In *CAV 2008*, Aarti Gupta and Sharad Malik (Eds.). LNCS, Vol. 5123. Springer Berlin Heidelberg, 176–189.

[35] Stephen Prajna and Ali Jadbabaie. 2004. Safety Verification of Hybrid Systems Using Barrier Certificates. In *Proceedings of the 7th International Workshop on Hybrid Systems: Computation and Control HSCC*. 477–492.

[36] Stephen Prajna, Ali Jadbabaie, and George J. Pappas. 2007. A framework for worst-case and stochastic safety verification using barrier certificates. *IEEE Trans. Automat. Control* 52, 8 (2007), 1415–1429.

[37] Luca Pulina and Armando Tacchella. 2010. An Abstraction-Refinement Approach to Verification of Artificial Neural Networks. In *Computer Aided Verification*. 243–257.

[38] Rahul Sharma, Saurabh Gupta, Bharath Hariharan, Alex Aiken, Percy Liang, and Aditya V. Nori. 2013. A Data Driven Approach for Algebraic Loop Invariants. In *Programming Languages and Systems*. Springer Berlin Heidelberg, 574–592.

[39] Rahul Sharma, Aditya V. Nori, and Alex. Aiken. 2012. Interpolants as Classifiers. In *Computer Aided Verification*. Springer Berlin Heidelberg, 71–87.

[40] Christoffer Sloth, George J Pappas, and Rafael Wisniewski. 2012. Compositional safety analysis using barrier certificates. In *Proc. of the Hybrid Systems: Computation and Control (HSCC)*. ACM, 15–24.

[41] Andrew Sogokon, Khalil Ghorbal, Yong Kiam Tan, and André Platzer. 2018. Vector Barrier Certificates and Comparison Systems. In *Formal Methods*. 418–437.

[42] Xiaowu Sun, Haitham Khedr, and Yasser Shoukry. 2019. Formal verification of neural network controlled autonomous systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019*. 147–156.

[43] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane S. Boning, and Inderjit S. Dhillon. 2018. Towards Fast Computation of Certified Robustness for ReLU Networks. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*. 5273–5282.

[44] Weiming Xiang, Hoang-Dung Tran, and Taylor T. Johnson. 2017. Output Reachable Set Estimation and Verification for Multi-Layer Neural Networks. *CoRR* abs/1708.03322 (2017).

[45] Bai Xue, Martin Fränzle, and Naijun Zhan. 2019. Inner-Approximating Reachable Sets for Polynomial Systems with Time-Varying Uncertainties. *IEEE Trans. Automat. Control* (2019), 1–1.

[46] Bai Xue, Martin Fränzle, Hengjun Zhao, Naijun Zhan, and Arvind Easwaran. 2019. Probably Approximate Safety Verification of Hybrid Dynamical Systems. In *Formal Methods and Software Engineering*. Springer International Publishing, 236–252.

[47] Zhengfeng Yang, Min Wu, and Wang Lin. 2015. Exact Verification of Hybrid Systems Based on Bilinear SOS Representation. (2015), 19 pages.

[48] Xia Zeng, Wang Lin, Zhengfeng Yang, Xin Chen, and Lilei Wang. 2016. Darboux-type Barrier Certificates for Safety Verification of Nonlinear Hybrid Systems. In *Proceedings of the 13th International Conference on Embedded Software*. ACM, Article 11, 10 pages.