Towards Verification-Aware Knowledge Distillation for Neural-Network Controlled Systems

(Invited Paper)

Jiameng Fan*, Chao Huang†, Wenchao Li*, Xin Chen‡ and Qi Zhu†
*Boston University, †Northwestern University, ‡University of Dayton
jmfan@bu.edu, chao.huang@northwestern.edu, wenchao@bu.edu, xchen4@udayton.edu, qzhu@northwestern.edu

***Abstract*—Neural networks are widely used in many applications ranging from classification to control. While these networks are composed of simple arithmetic operations, they are challenging to formally verify for properties such as reachability due to the presence of nonlinear activation functions. In this paper, we make the observation that Lipschitz continuity of a neural network not only can play a major role in the construction of reachable sets for neural-network controlled systems but also can be systematically controlled during training of the neural network. We build on this observation to develop a novel verification-aware knowledge distillation framework that transfers the knowledge of a trained network to a new and easier-to-verify network. Experimental results show that our method can substantially improve reachability analysis of neural-network controlled systems for several state-of-the-art tools.**

I. Introduction

Neural networks [10] are used in a wide variety of applications, such as image classification [11], natural language processing [20], and control of autonomous agents [22]. Despite their growing popularity, there are reservations about incorporating these nonlinear function approximators into safety-critical systems. One particular challenge is the verification problem of neural-network controlled systems (NNCSs), where a neural network is used as the controller in a closed-loop control system [15].

Several approaches have been proposed recently to reason about the reachability of NNCSs [15, 17, 7]. All of them rely on computing sound overapproximation of the reachable sets by leveraging specific nonlinearities of the neural networks, such as piecewise linearity for ReLU networks (networks with ReLU activation units) and differentiability for tanh networks. The first contribution of this paper is an observation that the treatment of nonlinearity can play an important role in the tightness of the overapproximations and/or computation time. We discuss this observation later in detail with an illustrative example. The second observation is that it is possible to control the "degree of nonlinearity" when training a neural network. The degree of nonlinearity that we consider in this paper is the Lipschitz constant of a neural network. Armed with these two observations, we propose a novel technique that extracts an easier-to-verify neural network from the original neural network while preserving control performance.

Specifically, we consider the framework of *knowledge distillation* [1, 12] as a way to obtain a new neural network that imitates the original network's behaviors but avoids large overapproximation errors. Knowledge distillation is the idea that after a cumbersome model has been trained, we can use a different kind of training to transfer the knowledge of this cumbersome model into a more structured or simpler model. In our case, we are more concerned with the ease of verification than the size of the model (although these two are related in certain cases). In the true spirit of knowledge distillation, we do not assume access to the original training data. This allows our method to be generally applied to different settings with how the original neural network might be trained, such as using reinforcement learning or supervised learning.

It has been shown that neural networks with many different types of activation functions are indeed Lipschitz continuous [16]. The magnitude of the Lipschitz constant of a neural network can substantially influence its effectiveness for a variety of tasks. For classification, the Lipschitz constant affects the robustness of a neural network to adversarial attacks [26]. In reinforcement learning, the Lipschitz constant of a neural-network controller can be upper bounded to guarantee stability [18] or plays an important role in constructing the stability certificate [3]. For verification of NNCSs, ReachNN [15] relies on the Lipschitz constant to conduct error analysis – smaller Lipschitz constant leads to tighter overapproximation and shorter verification time. While we focus on reducing the Lipschitz constant of the distilled neural network in this paper, we note that one can in fact replace Lipschitz constant with other verification-related metrics in our framework.

The joint consideration of training error and the value of Lipschitz constant naturally gives rise to a two-objective optimization problem. The first objective is to learn the input-output mapping of the original network, which can also be viewed as a regression problem. The second objective is to reduce the Lipschitz constant of the distilled network. To balance these two objectives, we develop a two-objective gradient descent method to simultaneously reduce the regression error and the Lipschitz constant of distilled network. We describe this method in detail in Sec. IV. In Sec. V, we present a case study on a neural-network controlled autonomous vehicle with a reach-avoid requirement. We show that reachability analysis can be significantly improved for the distilled network compared to using the original network.

Our work can be viewed as a paradigm shift from the traditional approach where verification is considered only after the design is done. For the verification community, this also opens up the possibility of reducing verification complexity by influencing how a system is trained. To the best of our knowledge, this is the first work that explicitly guides the training process by incorporating verification knowledge for reasoning about NNCSs.

A. Motivating Examples

We now present several examples that highlight the importance of Lipschitz constant for verifying NNCSs. These examples motivate the development of our verification-aware

knowledge distillation approach. Consider the following nonlinear control system [8]:

$$\dot{x}_1 = x_2, \quad \dot{x}_2 = ux_2^2 - x_1$$

where $x = (x_1, x_2)$ is the state variable and u is the control input that is computed by a neural-network controller κ, namely $u = \kappa(x)$. We want to check if the system will reach the target region $[0, 0.2] \times [0.05, 0.3]$ from anywhere in the initial set $[0.8, 0.9] \times [0.5, 0.6]$.

To study the impact of Lipschitz constant on verification of NNCSs, we first train multiple neural networks with different Lipschitz constants. Specifically, we train three neural networks with a large spread on Lipschitz constants for each type of ReLU/tanh/ReLU-tanh networks mimicking the same optimal controller. We consider three different reachability analysis tools to verify NNCSs, namely ReachNN, Verisig, and Sherlock. Since certain tools are designed to handle neural networks with specific types of activation functions, we use ReachNN to verify ReLU-tanh networks, Verisig to verify tanh networks, and Sherlock to verify ReLU networks. We note that ReachNN can in fact verify all three types of networks while Sherlock or Verisig can only handle one type.

Fig. 1 shows the result of the reachability analysis by different verification tools. We can observe that, for each tool, when the Lipschitz constant of the neural network controller grows, the reachable set expands more quickly. In some cases, the verification tools terminate due to either uncontrollable approximation error (Fig. 1f) or excessively long computation time (Fig. 1c). Interestingly, while a large Lipschitz constant can have a big impact on ReachNN and Verisig, it does not result in significant growth of the reachable sets computed by Sherlock. In the case of ReachNN, the estimation of the overapproximation error utilizes a sampling-based strategy and directly depends on the Lipschitz constant of the network. For Verisig, even though it does not make any explicit use of the Lipschitz constant of the network, results indicate that the Lipschitz constant has an indirect yet significant impact on the precision of its reachable set computation. On the other hand, the results with Sherlock show that the blowup in reachable sets as in the case of ReachNN and Verisig should be attributed to the change in the controller (due to a larger Lipschitz constant) and is in fact primarily caused by the construction of the overapproximation. These observations motivate us to train neural networks with smaller Lipschitz constants so that the resulting networks are easier to verify while maintaining control performance (measured via regression error with respect to an original network).

II. Related Work

A. Verification of NNCSs

When deploying neural-network controlled systems (NNCSs) in safety-critical scenarios, it is crucial to ensure that the unsafe states are unreachable. Tuncali et al. [28] propose the use of barrier certificates for the safety verification of NNCSs. In this paper, we consider more general reach-avoid specifications, where the system is required to reach a target region while avoiding unsafe regions. A key element in reachability analysis of NNCSs is the characterization of behaviors of the neural-network controller. For instance, output range analysis provides a straightforward characterization of a neural network in terms of a guaranteed range of the output given a set of inputs [19, 25, 29]. However, output range itself does not capture the actual input-output mapping. As a result, simply using output range to approximate the neural-network controller in a closed-loop system can lead to highly conservative results [7]. This thus motivates the use of a more tractable functional model (e.g. polynomials) to approximate the input-output mapping instead of the aforementioned set-based approximation. State-of-the-art tools include Sherlock [7], Verisig [17] and ReachNN [15]. Since this functional model is an approximation of the true neural network behaviors, the challenge is how to appropriately bound the error of the approximation. We discuss these three reachability tools in detail below.

Verisig leverages an interesting insight that differentiable activation functions can be equivalently transformed into differential equations. By applying this transformation for every neuron, a neural network is then transformed into a hybrid automaton and the transformed system can be verified by existing techniques [5, 14]. One limitation of Verisig is that, since differentiability of the activation functions is required, it cannot handle ReLU networks.

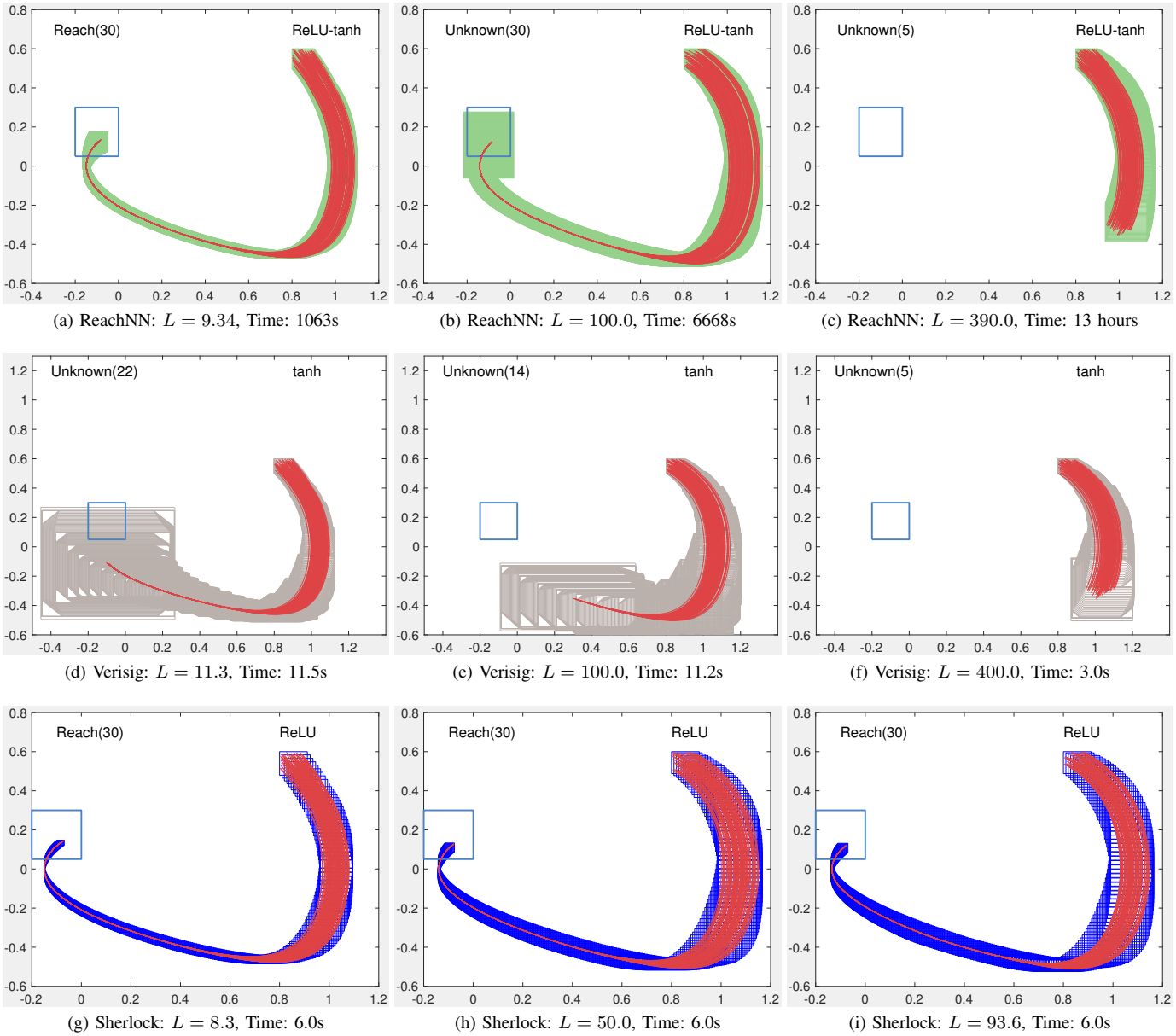
Sherlock targets ReLU networks which can be viewed as piecewise linear functions and computes overapproximation of reachable sets in a flowpipe-construction manner. It uses the Flow* tool [5] to compute the flowpipes (i.e. a set of states reachable by continuous dynamics from an initial set within a given time interval) under continuous dynamics and over-approximates the neural network by a regressive polynomial along with a remainder interval which is computed by solving a mixed-integer linear program.

ReachNN is more general than Verisig and Sherlock in the sense that it can handle any Lipschitz continuous network. At the core, it uses Bernstein polynomials to approximate the neural-network controller. On the flip side, without using more specific nonlinearity information of the target network, Huang et al. [15] resorts to a sampling-based approach to estimate the approximation error which is in turn based on the Lipschitz constant of the network. When the Lipschitz constant is large, it requires a large number of samples to perform the error estimation, which is time-consuming.

Motivated by the examples in Sec. I-A, where the Lipschitz constant of a neural network clearly plays a role in the precision of the reachable set estimation, this paper explores the direction of leveraging this information to retrain an existing neural network into one that has similar performance but with a Lipschitz constant that is as small as possible.

B. Knowledge Distillation

The particular framework that we use in this paper for retraining a neural network is known as *knowledge distillation*. There is a long history of work with ideas of distillation [6, 4, 1]. In its modern incarnation, Hinton et al. [12] formulates the idea of *knowledge distillation* as compressing the knowledge of an ensemble of models into a single model. The idea also extends to procedures such as the distillation of large but easy-to-train networks into small but harder-to-train networks [24], transfer from one architecture to another [9], integration with first-order logic [13] or other prior knowledge [30], and has been used in defense against adversarial attacks [23] and improving training stability [27]. The common theme



(a) ReachNN: $L = 9.34$, Time: 1063s (b) ReachNN: $L = 100.0$, Time: 6668s (c) ReachNN: $L = 390.0$, Time: 13 hours

(d) Verisig: $L = 11.3$, Time: 11.5s (e) Verisig: $L = 100.0$, Time: 11.2s (f) Verisig: $L = 400.0$, Time: 3.0s

(g) Sherlock: $L = 8.3$, Time: 6.0s (h) Sherlock: $L = 50.0$, Time: 6.0s (i) Sherlock: $L = 93.6$, Time: 6.0s

Fig. 1: Flowpipes computed by ReachNN, Verisig and Sherlock for different types of networks with different Lipschitz constants L. ReachNN, Verisig and Sherlock are used to verify on ReLU-tanh, tanh and ReLU neural networks respectively. Red curves denote the trajectories of (x_1, x_2) of the system simulated from randomly sampled states within the initial set. Blue rectangles represent the target region. Green rectangles are the flowpipes produced by ReachNN, gray rectangles are the flowpipes computed based on Verisig, and deep blue rectangles are the flowpipes computed using Sherlock.

behind this body of work is to use high-performing DNNs or prior knowledge to guide the training of shallower or more structured models.

Some recent work has also explored the idea of using *knowledge distillation* to circumvent the high complexity of the original networks in verification. Bastani et al. [2] propose to extract decision tree models from trained neural networks in the setting of reinforcement learning. Though decision tree policies are easier to verify, their application is limited to controllers with discrete outputs. In this paper, we consider neural network controllers whose outputs are continuous and the direction of transferring knowledge from a hard-to-verify network to an easy-to-verify network without losing accuracy.

III. Preliminaries

In this paper, we will mainly focus on the problem of reachability in neural-network controlled systems. We consider feed-forward neural networks (specifically multilayer perceptrons) as controllers for NNCSs. A feed-forward neural network, κ consists of $k-1 > 0$ hidden layers, and in each layer, there are $N>0$ neurons. Specifically, κ is given by

$$\kappa(\cdot) = \kappa_k(\kappa_{k-1}(\cdots \kappa_1(\cdot; W_1, b_1); W_2, b_2); W_k, b_k)$$

where W_i and b_i for $i = 1, 2, \ldots, k$ are learnable parameters as linear transformations connecting two consecutive layers and κ_i represents an element-wise nonlinear activation function following the linear transformations. We focus on three commonly used activation functions, ReLU, tanh and sigmoid. We define the type of closed-loop systems that use neural networks as the controllers as follows.

Definition 1 (Neural-Network Controlled Systems [15]). *A neural-network controlled system (NNCS) is denoted by a tuple $(\mathcal{X}, \mathcal{U}, F, \kappa, X_0)$, where $\mathcal{X}$ denotes the state space whose dimension is the number of state variables, $\mathcal{U}$ denotes the control input space whose dimension is the number of control inputs, F defines the continuous dynamics $\dot{x} = f(x, u)$, $\kappa : \mathcal{X} \to \mathcal{U}$ defines the input/output mapping of the neural-network controller, and $X_0 \subseteq \mathcal{X}$ denotes the initial state set.*

The function f is assumed to be Lipschitz continuous in x and continuous in u. Thus, NNCS is deterministic and unique solution to the dynamics exists for some time horizon $[0, T_{max})$ for given control inputs. The forward flowmap of an NNCS $(\mathcal{X}, \mathcal{U}, F, \kappa, X_0)$ is a function $\varphi_f : X_0 \times \mathbb{R}_{\geq 0} \to \mathcal{X}$. The function represents the evolution of NNCS starting from x_0 with fixed control to $\varphi_f(x_0, t)$ at the time t.

Definition 2 (Reach-Avoid Problem). *Given a NNCS $\mathcal{A}$, an initial set X_0, a target set X_{target}, an avoid set X_{avoid}, and a bounded time horizon $[0, T]$. We ask whether all executions of $\mathcal{A}$ starting from X_0 will reach X_{target} while avoiding X_{avoid} in the time interval $[0, T]$.*

Several state-of-art verification tools [7, 17, 15] rely on computing overapproximation of the exact reachable set to solve this reach-avoid problem. Thus, the ability to prove a reach-avoid property fundamentally depends on the quality of the overapproximation.

Definition 3 (Lipschitz Continuity). *A real-valued function $f : X \to \mathbb{R}$ is called Lipschitz continuous over $X \in \mathbb{R}^n$, if there exists a non-negative real number L, such that for any $x, x' \in X$:*

$$\|f(x) - f(x')\| \leq L\|x - x'\|$$

We compute the Lipschitz constant by using the upper bound estimation method from Ruan et al. [25]. The computation is done as follows.

Lemma 1 (Lipschitz constant for sigmoid/tanh/ReLU [25, 26]). *Fully connected layers with the sigmoid activation function $\mathcal{S}(Wx + b)$, tanh activation function $\mathcal{T}(Wx + b)$, and ReLU activation function $\mathcal{R}(Wx+b)$ have $\frac{1}{4}\|W\|$, $\|W\|$, $\|W\|$ as their Lipschitz constants, respectively.*

As we have seen in Fig. 1, neural networks with large Lipschitz constants can be challenging for verification tools to control the overapproximation error. ReachNN [15] directly use value of the Lipschitz constant in its error analysis. The basic idea of ReachNN is to approximate the neural network with Bernstein polynomials and bound the approximation error by using the following approach.

Theorem 1 (S-Error Estimation [15]). *Assume κ is a Lipschitz continuous function of $x = (x_1, \cdots, x_m)$ over $X = [l_1, u_1] \times \cdots \times [l_m, u_m]$ with a Lipschitz constant L. Let $P_{\kappa,d}$ be the Bernstein polynomial approximation of κ with the degree $d = (d_1, \cdots, d_m) \in \mathbb{N}^m$, and $X_c = \{c_k\}_{0 \leq k \leq p-1}$ be the sampling set with any given positive integer vector $p = (p_1, \cdots, p_m)$. Then the error bound $\bar{\varepsilon}_s(p)$ can be computed as*

$$\bar{\varepsilon}_s(p) = L\sqrt{\sum_{j=1}^{m}\left(\frac{u_j - l_j}{p_j}\right)^2} + \max_{0 \leq k \leq p-1} \|P_{\kappa,d}(c_k) - \kappa(c_k)\|. \quad (1)$$

The first term on the right side of Eq. 1 shows that the Lipschitz constant L is directly related to the error bound. If L is large, to reduce the first term we will need more samples (i.e. larger number of partition p_j, where $j = 1, \ldots, m$) which also increases computation time. On the other hand, if L is small, then less samples might be needed to keep the error small. This relationship motivates the use of *knowledge distillation* to retrain the original neural-network controller to reduce its Lipschitz constant.

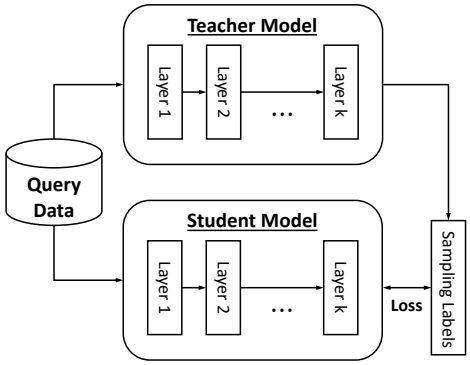


Fig. 2: Teacher-Student Paradigm

IV. Methods

We propose an iterative algorithm that retrains a neural network such that it simultaneously reduces the regression error with respect to the original input-output mapping and the Lipschitz constant of the resulting neural network.

A. Teacher-Student Paradigm

For more general applicability, we do not assume access to the original training data since the neural-network controller could have been trained using very different methods, such as reinforcement learning or MPC-guided training. We consider a teacher-student paradigm, where the original network acts as the *teacher*, and we aim to train a new neural network (i.e. the *student*) that can match the teacher's behaviors. During the retraining process, the *student* can query the *teacher* for "knowledge", which are input-output pairs in this case. Figure 2 illustrates the teacher-student paradigm.

Here, the *teacher* is the trained feed-forward neural network mapping from state space to control input space, $\kappa_{teacher}(\cdot) : x \mapsto u$, where the parameters are unknown. The *student* is represented by another feed-forward neural network parameterized by θ, $\kappa(\cdot; \theta) : x \mapsto u$. To eliminate effects caused by choosing different network architectures, we assume the structure of the *student* is the same as that of the *teacher*. In addition, using the same architecture makes it possible to make the regression error arbitrarily small.

B. Two-Objective Gradient Descent

At the high level, the retraining problem can be viewed as an optimization problem with two objectives. One is minimizing the regression error of input-output mappings between the original network and the retrained network, $J_{loss}(\theta)$. The other is minimizing a verification-related loss function, $J_{lip}(\theta)$. In this case, $J_{lip}(\theta)$ represents difference between the current value of the Lipschitz constant and some target value, L_{target} (which may not be known *a priori*). A straightforward way to optimize these two objectives would be to combine them into a single objective by taking a weighted sum. This is similar to using a regularization term top penalize large Lipschitz constants. The drawback of this approach is that it is difficult to tune the weights or know when it is possible to further reduce the Lipschitz constant without increasing the regression error.

We propose a two-objective gradient descent method inspired by *Task-Novelty Bisector* [31] to simultaneously reduce the regression error and the Lipschitz constant. We formulate the loss functions $J_{loss}(\theta)$ and $J_{lip}(\theta)$ as follows.

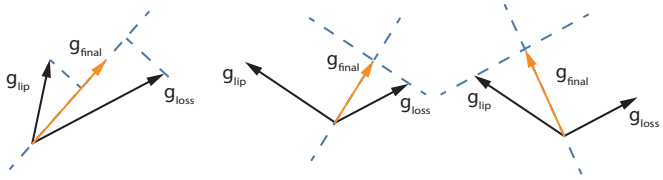
$$J_{loss}(\theta) = \mathbb{E}_{x\sim\mathcal{X}}\left[(\kappa(x;\theta) - \kappa_{original}(x))^2\right] \quad (2)$$

$$J_{lip}(\theta) = (L_\theta - L_{target})^2 \quad (3)$$

Their gradients with respect to the new network parameters are denoted as:

$$g_{loss} = \frac{\partial J_{loss}}{\partial \theta} \quad g_{lip} = \frac{\partial J_{lip}}{\partial \theta} \quad (4)$$

Taking a weighted sum of the two objective functions is similar to taking the weighted average of the corresponding gradients g_{loss} and g_{lip}. However, similar to the problem of weight tuning, the weighted average could result in biased gradient descent on the two objectives. In our case, it will be either losing regression accuracy or making the network hard to verify. To address this issue, we update the new network in the direction of the angular bisector of the two gradients and average the projected vectors of the two gradients on this direction if $g_{loss}\cdot g_{lip}>0$ as illustrated in Fig. 3a. This results in an update that is expected to reduce both the regression error and the Lipschitz constant. If $g_{loss}\cdot g_{lip}<0$, we will update along a gradient that is the projected vector of g_{loss} onto the hyperplane perpendicular to g_{lip}, as shown in Fig. 3b. This gradient prioritizes reducing the regression error over seeking for smaller Lipschitz constant and avoids increasing $J_{lip}(\theta)$ explicitly.



(a) $g_{loss}\cdot g_{lip}>0$ (b) $g_{loss}\cdot g_{lip}<0$ (c) $g_{loss}\cdot g_{lip}<0$ (reprioritization)

Fig. 3: Final update gradient that improves both objectives

C. Gradient Reprioritization Scheme

For general neural networks, the two-objective gradient descent approach cannot guarantee that the Lipschitz constant will not increase when $g_{loss}\cdot g_{lip}<0$. It is also possible that the case of $g_{loss}\cdot g_{lip}>0$ will not appear later in the iterations. Hence, there are two options for this two-objective optimization problem. One is to specify a target Lipschitz constant and search for a network with the smallest regression error. The other is to specify the regression error bound and search for a network with the smallest Lipschitz constant.

In general, it is difficult to know *a prior* how small the Lipschitz constant or the regression error need to be. Nevertheless, the system that we are verifying at the end uses the retrained network instead of the original network as the controller. This means our approach is viable as long as we can prove reachability of the resulting system. Thus, we consider searching for network with the smallest Lipschitz constant that satisfies a pre-specified regression error bound or with the smallest Lipschitz constant such that reducing the regression error will increase the Lipschitz constant. The second case also implies that the gradient should be zero or vanishing for this two-objective optimization.

To further balance the gradients of two objectives, we propose to *reprioritize* the gradient direction to g_{lip} when $g_{loss}\cdot g_{lip}<0$ and the regression loss is less than some error bound ε. Instead of projecting g_{loss} onto the hyperplane perpendicular to g_{lip}, we prioritize reducing the Lipschitz constant. As shown in Fig. 3c, the final gradient takes the projection of g_{lip} onto the hyperplane that is perpendicular to g_{loss}, $g_{final}=g_{lip}-\frac{g_{lip}\cdot g_{loss}}{\|g_{loss}\|}g_{loss}$. On one hand, if this gradient does not increase the regression error, we can further reduce the Lipschitz constant without increasing the regression error. On the other hand, if this gradient results in a larger regression error, we can follow the original two-objective gradient descent rule to reduce the regression error. In a nutshell, this reprioritization scheme addresses the issue where the vanilla two-objective gradient descent approach constantly reduces the regression error after satisfying the regression error bound but neglects to reduce the Lipschitz constant.

To speed up the retraining process and avoid getting stuck at local optima, we use simulated annealing to inject stochasticity when choosing the regression loss error bound ε_{loss} in the neighborhood of ε. Algorithm 2 formalizes this procedure.

D. Two-Objective Knowledge Distillation

We summarize our algorithm below. By combining the two-objective gradient descent and simulated annealing, our method retrains a new network with a smaller Lipschitz constant while preserving the (control) performance of the original network. The process terminates when the final gradient is vanishing, i.e. $g_{loss}\cdot g_{lip}=-1$ or $\|g_{loss}\|\|g_{lip}\|=0$, or when the Lipschitz constant cannot be reduced further with our reprioritization scheme, or when the two gradients agree with each other. For the first case, it means that taking any gradient direction will negatively impact at least one of the objectives. Algorithm 1 describes our proposed method formally.

V. Case Study

In this section, we present a case study involving an autonomous driving task. We consider a simple Dubins car model, and a neural-network controller that controls the car to navigate around static obstacles to reach a target region. For convenience, the original neural network is trained from an optimal controller that satisfies the reach-avoid specification. We first describe the system dynamics. Then, we describe the training and retraining process. Finally, we show the proposed

Algorithm 1 Two-Objective Knowledge Distillation

Input: State space $\mathcal{X}$, sample size N, target Lipschitz constant L_{target}, error bound ε, learning rate α.
Initialize: New network parameters θ, regret loss $= +\infty$, Lipschitz constant reducing list L_s, $\varepsilon_{loss} = \varepsilon$, reduce$_{\text{lip}}$ = False, stop = False
while loss $> \varepsilon$ **do**
  Sample N state samples from $\mathcal{X}$ as $x_1, \ldots, x_N$
  loss $= \sum_{i=1}^{N}((\kappa(x_i;\theta) - \kappa_{original}(x_i))^2)$
  Compute the Lipschitz constant L_θ by Lemma. 1.
  if reduce$_{\text{lip}}$ **then**
    Append L_θ to L_s
    if $L_\theta <$ min(L_s) **then**
      stop = True
    end if
  end if
  Compute the gradients g_{loss} and g_{lip}
  if $g_{loss} \cdot g_{lip} > 0$ and not stop **then**
    $g_{dir} = \frac{bisector(g_{loss}, g_{lip})}{\|bisector(g_{loss}, g_{lip})\|}$
    $g_{final} = \left(\frac{g_{loss}+g_{lip}}{2} \cdot g_{dir}\right) \cdot g_{dir}$
    reduce$_{\text{lip}}$ = True
  else
    if $g_{loss} \cdot g_{lip}$==-1 or $\|g_{loss}\|\|g_{lip}\|$==0 **then**
      break – Termination
    else if stop and loss$<\varepsilon$ **then**
      break – Termination
    end if
    Compute new ε_{loss} using Algorithm 2.
    if loss $> \varepsilon_{loss}$ or stop **then**
      $g_{final} = g_{loss} - \frac{g_{loss} \cdot g_{lip}}{\|g_{lip}\|} g_{lip}$
    else if $L_\theta > L_{target}$ **then**
      $g_{final} = g_{lip} - \frac{g_{lip} \cdot g_{loss}}{\|g_{loss}\|} g_{loss}$
      loss $= +\infty$, reduce$_{\text{lip}}$ = True
    end if
  end if
  $\theta = \theta + \alpha \cdot g_{final}$
end while

knowledge distillation process improves the effectiveness of the verification tools.

A. Experiment Setup

The continuous dynamics of a Dubins car is defined by

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = u$$

where (x, y) is the coordinate of the car and represents the car's position in a 2D plane. The variable θ is the angle between the car's heading orientation and the x-axis. We denote v as the speed of the car and $\dot{\theta}$ as the turning rate which we will refer to as *steering control* u. In our experiments, we assume that car speed, v, is constant.

The car is placed in a corridor environment, where obstacles consist of walls and a square obstacle represented by yellow lines in Fig. 4. The car starts from a location in the initial set $[-4.5, -4.4]\times[0, 0.1]\times[0, 0.1]$. Its *steering control* is bounded in the interval $[-1, 1]$ and we set car speed as v=1. The target region lies in $[1.5, 2.25]\times[-3.5, -2.75]\times[-\pi, \pi]$. The overall

Algorithm 2 Simulated Annealing for Error Bound Selection

Input: Error tolerance ε, Error Bound ε_{loss}, Iteration number k, Initial Temperature T_0, Factor $\beta > 0$ and loss
$\varepsilon_{new} \sim$ *Uniform*$(0.5\varepsilon, 1.5\varepsilon)$
if loss $< \varepsilon_{new}$ **then**
  return ε_{new}
else
  $T = \frac{T_0}{1+\beta k}$
  $P = exp(-\frac{1}{T})$
  if $P >$ *Uniform*$(0, 1)$ **then**
    return ε_{new}
  else
    return ε_{loss}
  end if
end if

control goal is to drive the car to the target region without colliding with the obstacles.

B. Training of the Original Neural Network

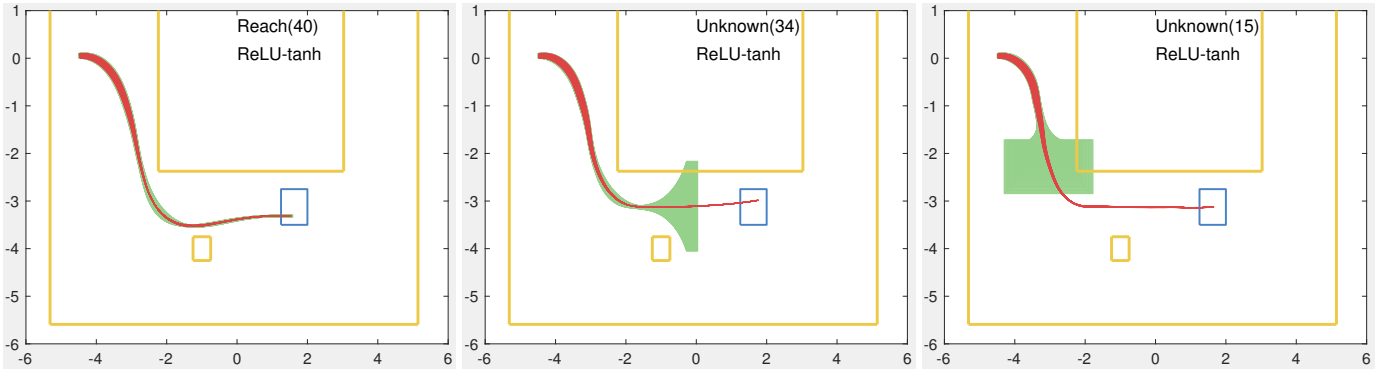
We used a feed-forward neural network with two hidden layers and 20 neurons in each hidden layer as the controller. The network has ReLU activation function for hidden layers and tanh activation function for output layer. It takes state vector (x, y, θ) as inputs and outputs *steering control* u. We used the Level Set Toolbox [21] to compute a value map over state-control space and derive the optimal control that will keep the car away from obstacles at all times and reach the target region at the 40$^{\text{th}}$ time step.

We built the training set by randomly sampling states from the state space and collecting the corresponding optimal controls. The training set consists of 1,300,000 state-control pairs. The original network was trained from this training set with squared loss and stochastic gradient descent. In practice, such a neural-network controller might be trained from a variety of methods or by a third party. Thus, we assumed that we did not have access to this training data.

C. Retraining and Verification

After training the initial neural network, we used ReachNN to check the reach-avoid property. ReachNN reports *Unknown* since the overapproximation of the reachable set intersects with avoid set at the 15$^{\text{th}}$ step. However, when we simulated the controller from randomly chosen states in the initial set, we observed that the controller can successfully meet the reach-avoid objective. The simulated trajectories are shown in red in Fig. 4c. It turns out that the original neural-network controller has a large Lipschitz constant 244.3, which contributes to the large overapproximation error.

We retrained two networks from the original neural network with two different L_{target} values 100.0 and 0.0. For the other hyperparameters, we set the error tolerance ε to 0.05, learning rate α to 10^{-3}, and queried 100,000 samples from the original neural-network controller at each iteration. For $L_{\text{target}} = 100.0$, we were able to obtain a network with a Lipschitz constant of 100.0 after distillation. For $L_{\text{target}} = 0.0$, we obtained a network with a small Lipschitz constant 13.1. We verified the system with these two neural-network controllers separately using ReachNN and the results are shown in Fig. 4.



(a) ReachNN: $L = 13.1$, Time: 4657s (b) ReachNN: $L = 100.0$, Time: 13765s (c) ReachNN: $L = 244.3$, Time: 13 hours

(d) Verisig: $L = 17.1$, Time: 12s (e) Verisig: $L = 150.4$, Time: 12s (f) Verisig: $L = 348.6$, Time: 15s

Fig. 4: Reachability analysis results: Yellow lines represent boundaries of the obstacles and form the avoid set. Blue square represents the target region. Three ReLU-tanh neural-network controllers with different Lipschitz constants are checked using ReachNN ((a) - (c)). Similarly, three tanh neural-network controllers are checked using Verisig ((d) - (f)). Note that while it appears that Verisig terminates when the overapproximation is still tight, by monitoring the execution of Verisig, we found the width of the remainder that represents the approximation error would explode in the following control step.

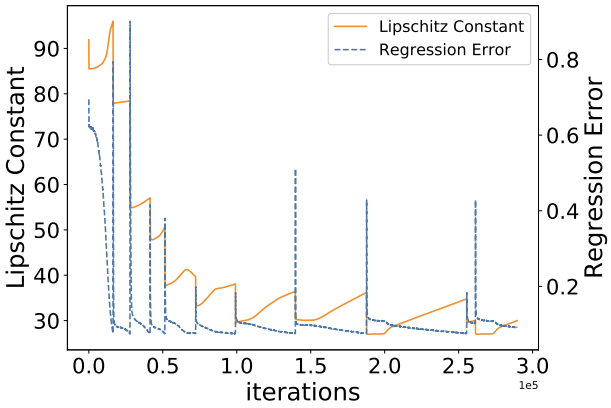


Fig. 5: Evolution of Lipschitz constant and regression error during the distillation process for the network used in Fig. 4a.

In Fig. 4a, we can see that the neural network with a smaller Lipschitz constant turns the car more slowly. As a result, the car ends up being closer to the square obstacle compared to a neural network with a larger Lipschitz constant. On the other hand, even though the three neural-network controllers do not have identical behaviors or performance, sampled simulation trajectories indicate that they may all satisfy the reach-avoid specification. The reachable sets computed by ReachNN ended up being very different for the three NNCSs. For the network with $L = 100.0$, the reachable set blows up and intersects with the wall before it reaches the target region at the 34^{th} step. For the network with the smallest Lipschitz constant $L = 13.1$, ReachNN was able to show that the reachable set is contained in the target set at the 40^{th} step and never intersects with the obstacle regions. This result clearly shows the benefit of reducing the Lipschitz constant of the neural-network controller via our distillation approach in reachability analysis.

We also evaluated Verisig in the same manner and the results are shown in Fig. 4d, 4e and 4f. We first trained a tanh neural-network controller similar to the setting in Sec. V-B and retrained two new tanh networks with the same ε but different L_{target}s. For the controllers with larger Lipschitz constants, Verisig terminates earlier and reports `unknown` as the verification result. This could be because a larger Lipschitz constant causes more neurons to activate in the network for the same inputs, which in turn results in constructing a larger equivalent hybrid automaton which is more difficult to handle for reachability analysis.

Fig. 5 illustrates the evolution of Lipschitz constant and

regression error during the distillation process for training the controller in Fig. 4a. The values were sampled every 100 iterations. We can observe that the Lipschitz constant decreases over time but both the Lipschitz constant and the regression error fluctuate. The fluctuations in fact follow a certain pattern that reflects the effect of our two-objective gradient descent approach and reprioritization scheme. A decrease in the regression error is often coupled with an increase in the Lipschitz constant. When the regression error is small enough, it triggers the reprioritization scheme and our algorithm switches to focusing on reducing the Lipschitz constant, which in turn may cause an increase in the regression error. Eventually, our algorithm returned a network with a small Lipschitz constant and a small regression error with respect to the original network.

VI. Conclusion

In this paper, we present a novel technique to distill a trained network to produce a new neural network that is more verification friendly. The key idea is to modify the usual gradient descent to balance the two objectives of reducing the regression error of retraining and reducing the Lipschitz constant of the target network. We show that our technique enables state-of-the-art verification tools to tighten the overapproximation of reachable set computations and also reduce their computation time. In the future, we plan to explore related ideas such as model compression/pruning for verification of neural networks and NNCSs.

Acknowledgment

This work was supported in part by the DARPA BRASS program under agreement number FA8750-16-C-0043, NSF under award number 1834701, 1834324, 1839511, 1724341 and 1646497, and by the Air Force Research Laboratory (AFRL).

References

[1] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[2] O. Bastani, Y. Pu, and A. Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2494–2504, 2018.

[3] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause. Safe model-based reinforcement learning with stability guarantees. In *Advances in neural information processing systems*, pages 908–918, 2017.

[4] C. Bucilu, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM, 2006.

[5] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Proc. of CAV'13*, volume 8044 of *LNCS*, pages 258–263. Springer, 2013.

[6] M. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*, pages 24–30, 1996.

[7] S. Dutta, X. Chen, and S. Sankaranarayanan. Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *Hybrid Systems: Computation and Control (HSCC)*, pages 157–168. ACM Press, 2019.

[8] E. Gallestey and P. Hokayem. Lecture notes in nonlinear systems and control, 2019.

[9] K. J. Geras, A.-r. Mohamed, R. Caruana, G. Urban, S. Wang, O. Aslan, M. Philipose, M. Richardson, and C. Sutton. Blending lstms into cnns. *arXiv preprint arXiv:1511.06433*, 2015.

[10] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[12] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[13] Z. Hu, X. Ma, Z. Liu, E. Hovy, and E. Xing. Harnessing deep neural networks with logic rules. *arXiv preprint arXiv:1603.06318*, 2016.

[14] C. Huang, X. Chen, W. Lin, Z. Yang, and X. Li. Probabilistic safety verification of stochastic hybrid systems using barrier certificates. *TECS*, 16(5s):186, 2017.

[15] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *arXiv preprint arXiv:1906.10654*, 2019.

[16] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer, 2017.

[17] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. *arXiv preprint arXiv:1811.01828*, 2018.

[18] M. Jin and J. Lavaei. Stability-certified reinforcement learning: A control-theoretic perspective. *arXiv preprint arXiv:1810.11505*, 2018.

[19] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[20] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

[21] I. M. Mitchell. The flexible, extensible and efficient toolbox of level set methods. *Journal of Scientific Computing*, 35(2-3): 300–329, 2008.

[22] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[23] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

[24] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[25] W. Ruan, X. Huang, and M. Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*, 2018.

[26] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[27] Z. Tang, D. Wang, and Z. Zhang. Recurrent neural network training with dark knowledge transfer. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5900–5904. IEEE, 2016.

[28] C. E. Tuncali, H. Ito, J. Kapinski, and J. V. Deshmukh. Reasoning about safety of learning-enabled components in autonomous cyber-physical systems. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2018.

[29] W. Xiang and T. T. Johnson. Reachability analysis and safety verification for neural network control systems. *arXiv preprint arXiv:1805.09944*, 2018.

[30] R. Yu, A. Li, V. I. Morariu, and L. S. Davis. Visual relationship detection with internal and external linguistic knowledge distillation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1974–1982, 2017.

[31] Y. Zhang, W. Yu, and G. Turk. Learning novel policies for tasks. *arXiv preprint arXiv:1905.05252*, 2019.