Machine Learning (2023) 112:2903–2931
https://doi.org/10.1007/s10994-021-06065-9

Generating probabilistic safety guarantees for neural network controllers

Sydney M. Katz[1] · **Kyle D. Julian**[1] · **Christopher A. Strong**[2] · **Mykel J. Kochenderfer**[1]

Received: 22 September 2020 / Revised: 17 July 2021 / Accepted: 15 September 2021 /
Published online: 19 October 2021
© The Author(s), under exclusive licence to Springer Science+Business Media LLC, part of Springer Nature 2021

Abstract
Neural networks serve as effective controllers in a variety of complex settings due to their ability to represent expressive policies. The complex nature of neural networks, however, makes their output difficult to verify and predict, which limits their use in safety-critical applications. While simulations provide insight into the performance of neural network controllers, they are not enough to guarantee that the controller will perform safely in all scenarios. To address this problem, recent work has focused on formal methods to verify properties of neural network outputs. For neural network controllers, we can use a dynamics model to determine the output properties that must hold for the controller to operate safely. In this work, we develop a method to use the results from neural network verification tools to provide probabilistic safety guarantees on a neural network controller. We develop an adaptive verification approach to efficiently generate an overapproximation of the neural network policy. Next, we modify the traditional formulation of Markov decision process model checking to provide guarantees on the overapproximated policy given a stochastic dynamics model. Finally, we incorporate techniques in state abstraction to reduce overapproximation error during the model checking process. We show that our method is able to generate meaningful probabilistic safety guarantees for aircraft collision avoidance neural networks that are loosely inspired by Airborne Collision Avoidance System X (ACAS X), a family of collision avoidance systems that formulates the problem as a partially observable Markov decision process (POMDP).

Editors: Daniel Fremont, Alessio Lomuscio, Dragos Margineantu, Cheng Soon-Ong.

✉ Sydney M. Katz
smkatz@stanford.edu

Kyle D. Julian
kjulian3@stanford.edu

Christopher A. Strong
castrong@stanford.edu

Mykel J. Kochenderfer
mykel@stanford.edu

[1] Department of Aeronautics and Astronautics, Stanford University, Stanford, CA, USA

[2] Department of Electrical Engineering, Stanford University, Stanford, CA, USA

Keywords Neural network controller · Verification · Model checking · Safety

1 Introduction

Neural networks provide a means to represent complex control policies, making them particularly useful in complicated problem domains where an agent must make decisions over a large input space (Mnih et al. 2015). Recently, neural networks have been proposed as controllers in safety-critical applications such as aircraft collision avoidance and autonomous driving (Julian et al. 2016, 2019a; Bouton 2020; Pan et al. 2017). Using neural networks in these settings presents major challenges. Due to the inherent complexity and unpredictable nature of neural networks, they are difficult to certify for use in safety-critical applications. Performance in Monte Carlo simulations is not enough to guarantee that the network will provide safe actions in all scenarios. To this end, recent work in formal methods has resulted in tools for verifying properties of neural networks (Katz et al. 2017, 2019; Wang et al. 2018; Tjeng et al. 2017). Given a bounded input set, these tools provide guarantees on characteristics of the output set.

Using neural network verification tools to prove properties in this manner represents a step towards the ability to certify neural networks as safe; however, these works simply check input-output properties specified by a human designer without considering the closed-loop behavior of the system. In order to guarantee safety for neural network controllers, there is a need for a more principled approach to selecting the properties that a neural network must satisfy for safe operation. By taking into account a dynamic model, we can create a closed-loop system that describes the effect of the neural network controller's actions on its environment. We can then use this system to better understand what constitutes a "safe" neural network output. Previous work has evaluated the safety of the closed-loop system using various forms of reachability analysis (Julian and Kochenderfer 2019b, a; Huang et al. 2019; Xiang and Johnson 2018; Xiang et al. 2018, 2019; Dutta et al. 2019; Ivanov et al. 2019; Clavière et al. 2020; Lopez et al. 2021). One limitation of the reachability approaches used in previous work is their inability to properly account for stochasticity in the system dynamics, which is present in many real-world systems. For instance, a number of the approaches assume no uncertainty in the dynamics model when computing reachable sets (Clavière et al. 2020; Lopez et al. 2021). While other work takes into account this uncertainty by overapproximating the system dynamics, the binary nature of the output of this analysis does not properly reflect the stochastic nature of the dynamics model (Julian and Kochenderfer 2019a). In particular, this analysis simply flags states as reachable without specifying the likelihood of reaching them. Therefore, even if the probability of reaching an unsafe state is extremely low, this technique would mark the overall system as unsafe.

In this work, instead of determining solely whether unsafe states are reachable, we develop a method that takes as input a stochastic dynamics model and provides probabilistic safety guarantees on the closed-loop system. Similar to Julian and Kochenderfer (2019b), we divide the input space into small cells and run each input region through a neural network verification tool (Julian and Kochenderfer 2019a, b). Using the results of the neural network verification tool to define an action space, we formulate the closed-loop verification problem as a Markov decision process (MDP). This formulation allows us to draw upon techniques from MDP model checking to approximate the probability of

reaching an unsafe state from any particular cell given a probabilistic model of the dynamics (Baier and Katoen 2008; Lahijanian et al. 2011; Bouton et al. 2020; Bouton 2020).

We modify the model checking formulation to ensure an overapproximation of the probabilities and outline both online and offline methods to reduce overapproximation error. Specifically, we develop an adaptive verification method that addresses limitations mentioned in previous work to efficiently divide the input region into cells (Julian and Kochenderfer 2019a). We show that this method better approximates the decision boundaries of the neural network and processes the input space faster than a naïve approach to state space partition. We further reduce overapproximation error by using ideas from state abstraction to split safety-critical regions of the state space during the solving process (Munos and Moore 2002). Our contributions are summarized as follows.

- We show how to adapt techniques in MDP model checking to generate probabilistic safety guarantees on neural network controllers operating in environments with stochastic dynamics.
- We create a method to obtain an overapproximated neural network policy using existing neural network verification tools. Specifically, we introduce an adaptive verification approach to automatically partition the input space in a way that reduces overapproximation error.
- We show how to use techniques in state abstraction to reduce overapproximation error in the estimated probability during the model checking process.

We apply our method to aircraft collision avoidance neural networks and show that we can use it to provide meaningful safety guarantees on a neural network controller.

2 Background

In this work, we formulate the closed-loop verification problem as a Markov decision process (MDP) and use this formulation to apply techniques in probabilistic model checking to evaluate the safety of a neural network controller. This section outlines the necessary background on MDPs and probabilistic model checking.

2.1 Markov decision process

An MDP is a way of encoding a sequential decision making problem where an agent's action at each time step depends only on its current state (Kochenderfer 2015). An MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T(s, a, s')$ is the probability of transitioning to state s' given that we are in state s and take action a, $R(s, a)$ is the reward for taking action a in state s, and γ is the discount factor. Using this formulation, we can solve for a policy π that maps states to actions. To do so, we define an action-value function $Q(s, a)$ that represents the discounted sum of expected future rewards when taking action a from state s. The optimal action-value function $Q^*(s, a)$ can be found using a form of dynamic programming called value iteration. Value iteration relies on iterative updates using the Bellman equation (Bellman 1952):

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a' \in A} Q^*(s',a') \tag{1}$$

We can extract the policy from the action-value function by simply choosing the action with the maximum value at state s:

$$\pi(s) = \arg\max_{a \in A} Q^*(s, a) \tag{2}$$

2.2 Probabilistic model checking

Probabilistic model checking for MDPs has been well studied and often involves determining the probability of satisfying a property expressed using Linear Temporal Logic (LTL) (Baier and Katoen 2008; Lahijanian et al. 2011; Bouton et al. 2020). An LTL formula consists of atomic propositions connected by logical or temporal operators (Baier and Katoen 2008). Our goal is to assign a probability $\Pr^{\pi}(s)$ of satisfying the LTL specification to each state $s \in \mathcal{S}$. For any LTL formula, this computation reduces to a reachability problem for a set of states $\mathcal{B}$ (Baier and Katoen 2008; Bouton 2020). In traditional MDP model checking, we seek to find the maximum probability of reaching states in $\mathcal{B}$ while following policy π from each state $s \in \mathcal{S}$. This probability can be written recursively as

$$\Pr^{\pi}(s) = \sum_{s' \in \mathcal{S}} T(s' \mid s, \pi(s)) \Pr^{\pi}(s') \tag{3}$$

for all states $s \notin \mathcal{B}$. All states $s \in \mathcal{B}$ are assigned a probability of one.

Equation (3) can also be written in a form that is analogous to the action-value function in Eq. (1) to represent the probability of satisfying the LTL formula when action a is taken from state s as follows

$$\Pr^{\pi}(s, a) = \sum_{s' \in \mathcal{S}} T(s' \mid s, a) \Pr^{\pi}(s', \pi(s')) \tag{4}$$

Noting the similarity between Eqs. (1) and (4), we can solve for the probabilities using value iteration. The problem reduces to solving for the value function of an MDP with a modified reward function to represent probabilities (Bouton et al. 2020; Bouton 2020). The immediate reward is one for being in a state in $\mathcal{B}$ and zero for being in any other state.

3 Approach

We assume that we are given a neural network controller that represents the value function for an MDP policy π, which maps points in a bounded input space $\mathcal{S}$ to an action in the action space $\mathcal{A}$. We also assume that we are given a stochastic dynamics model in the form of a transition model $T(s' \mid s, a)$ and a safety specification on the closed-loop system written in the form of an LTL formula. Using this information, our goal is to determine the probability that the neural network controller satisfies this specification from each state $s \in \mathcal{S}$. If the input space $\mathcal{S}$ were discrete, we could directly apply the technique outlined in Sect. 2.2 to determine these probabilities. However, because neural network controllers typically take in a continuous range of states, we must introduce approximations into the model checking process to handle the continuous input space.

We make these approximations by partitioning the input space $\mathcal{S}$ into a finite number of smaller regions called cells, $c \in \mathcal{C}$, and modifying the model checking process to work with cells rather than states. We break the problem into two steps. The first step involves using a

neural network verification tool to obtain an overapproximated neural network policy $\tilde{\pi}$ that operates on cells in $\mathcal{C}$. Using this policy, the second step uses probabilistic model checking to generate an overapproximated probability of reaching an unsafe state from each cell in $\mathcal{C}$. For both steps, we develop techniques to reduce overapproximation error in the estimated probabilities, which we outline in Sect. 4.

Figure 1 shows a simple example of a slippery continuum world that will be used to aid in our explanations of each step of our approach. In this example, the agent's objective is to reach a point within a set of goal states represented by the green region in the upper right corner without falling into a pit in the center of the world represented by the red region. The agent can select from four actions: up, down, left, or right. Because the world is slippery, taking an action results in a 70% chance of moving one unit in the specified direction and a 10% chance of moving one unit in each of the other directions. The plot on the right of Fig. 1 shows a sensible neural network policy for an agent in this world to follow along with a potential partition of the state space into cells. Our goal is to determine the overapproximated probability that an agent following this policy from each cell will fall into the pit.

3.1 Policy overapproximation

The first step in modifying traditional MDP model checking to use cells involves defining a policy that takes a cell as input rather than a single state. Because each cell contains multiple states, it is possible for cells to map to multiple actions. For example, the cell in the bottom left corner of the policy plot in Fig. 1 contains some states that map to the right action and others that map to the up action. For each cell $c \in \mathcal{C}$, we use a neural verification tool to obtain the possible actions $\mathcal{A}_c \subseteq \mathcal{A}$ that the neural network could output for some point in c. The results provide an overapproximated policy $\tilde{\pi}$ that maps a cell c to a subset of the action space $\mathcal{A}_c$ in contrast with π, which maps a specific state in the input space to a specific action. We assume that any point in c could yield any action in $\mathcal{A}_c$. Therefore, any cell that has multiple actions in $\mathcal{A}_c$ contributes to an overapproximation of the neural network policy. Policy overapproximation is the first source of overapproximation error in the probability estimate.

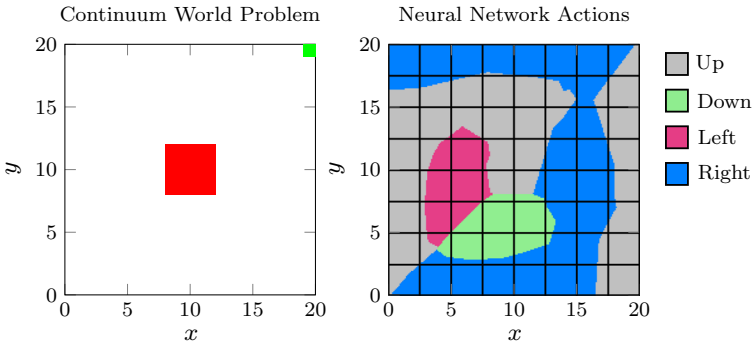

Fig. 1 Continuum world explanatory example. The plot on the left shows the setup of the continuum world. The goal of the agent is to reach a point in the green area while avoiding the red area. The plot on the right shows an example neural network policy for this problem (Color figure online)

3.2 Model checking

Once we have an overapproximated policy, we can further modify the model checking framework to determine the probability of satisfying the LTL safety specification from each cell $c \in \mathcal{C}$. We first convert the LTL specification to a reachability problem for a set of states $\mathcal{B}$ either directly or by using the methods in Baier and Katoen (2008). In the continuum world example, $\mathcal{B}$ is the region of the state space that corresponds to the pit. Next, we adapt Eq. (3) to determine the probabilities for each cell $c \in \mathcal{C}$ using our overapproximated neural network policy $\tilde{\pi}$ as

$$\Pr^{\tilde{\pi}}(c) = \max_{a \in \mathcal{A}_c} \sum_{c' \in \mathcal{C}} T(c' \mid c, a) \Pr^{\tilde{\pi}}(c') \tag{5}$$

where $\mathcal{A}_c$ uses the neural network verification results and contains the set of actions that could be taken in cell c. All cells that overlap with $\mathcal{B}$ are assigned a probability of one. The maximization in Eq. (5) corresponds to taking the worst-case action in $\mathcal{A}_c$.

The transition model, $T(c' \mid c, a)$, is modified to determine transitions between cells rather than states and to ensure that the resulting probabilities represent an overapproximation of the true probabilities. In this work, we restrict our approach to models in which taking an action results in a finite number of outcomes. We assume that taking action a from state s has n possible outcomes with probabilities according to $T(s' \mid s, a)$. For example, in the continuum world, these outcomes would be the result of moving one unit up, down, left, and right. Let p_i represent the probability of the ith outcome. Let $\mathcal{S}'_{1:n}$ represent regions of the state space that contain all possible next states from cell c for each outcome $i \in 1, \ldots, n$. We define $\mathcal{C}'_{1:n}$ as the sets of cells that overlap with $\mathcal{S}'_{1:n}$. In order to preserve the overapproximation in our probabilities, we assign all of the probability for outcome i to the worst-case cell in $\mathcal{C}'_i$ as follows

$$T(c' \mid c, a) = \sum_i \begin{cases} p_i, \text{ if } c' = \arg\max_{c'' \in \mathcal{C}'_i} \Pr^{\tilde{\pi}}(c'') \\ 0, \text{ otherwise} \end{cases} \tag{6}$$

Equation (6) preserves the overapproximation by assuming that *all* points in a cell transition to the worst-case next state realizable from *any* point in the cell. Figure 2 shows a visual representation of the transition model for the continuum world adapted for use with cells. All cells are labeled with their probability estimates, and the figure shows the result of taking the up action from the cell highlighted in black. The shaded regions represent $\mathcal{S}'_{1:4}$. The highlighted cells represent the worst-case cells that overlap each region $\mathcal{S}'_i$.

With these modifications in place, we can apply dynamic programming using Eq. (5) to determine the probability of reaching states in $\mathcal{B}$ for each cell $c \in \mathcal{C}$. Figure 3 shows the results of this process for two different cell partitions: a coarse partition (top row) and a fine partition (bottom row). In both cases, all cells that overlap with the pit have a probability of one assigned to them, and the probability of falling into the pit decreases as cells get further away from it. The coarse partitioning results in significantly more overapproximation error than the fine partitioning. However, partitioning the neural network input space uniformly into small cells significantly increases complexity, especially as the dimension of the input space increases. Sect. 4 describes ways to reduce overapproximation error that only require a fine resolution in critical areas of the state space.

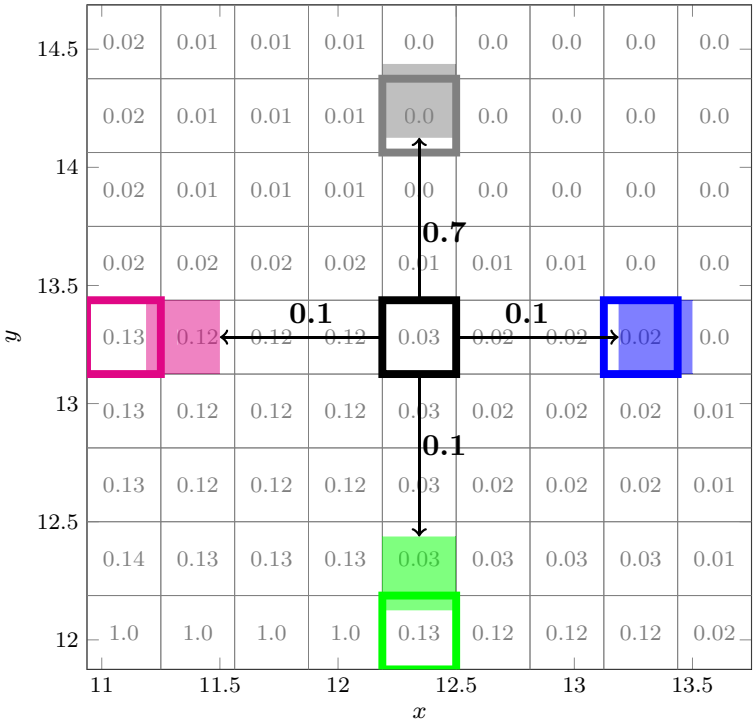

Fig. 2 Illustration of cell transitions for the continuum world example. Each cell is labeled with its corresponding probability estimate. Each shaded region shows the set of possible next states with their corresponding probabilities of being reached when the up action is taken from the cell highlighted in black. We assign all probability for each region to the worst-case cell that it overlaps with shown by the highlighted cells

4 Reducing overapproximation error

The model checking formulation presented here results in two sources of overapproximation error. The first source of error, which we will refer to as policy overapproximation error, is the overapproximation of the neural network policy from the neural network verification tools. We assume that the actions in $\mathcal{A}_c$ may be taken at any point in the cell and that we always take the action with the worst-case probability (see the maximization in Eq. (4)). Even if the worst-case action covers only a small portion of a cell, we must assume that the action is possible at any point in the cell.

The second source of error, which we will refer to as worst-case transition error, is the overapproximation in the transition model shown in Eq. (6). We assume that all points in the cell transition to the worst-case next cell for a given outcome. In order to produce meaningful probabilistic guarantees, it is crucial that we develop methods to reduce the overapproximation error. In this work, we present both offline and online error reduction methods.

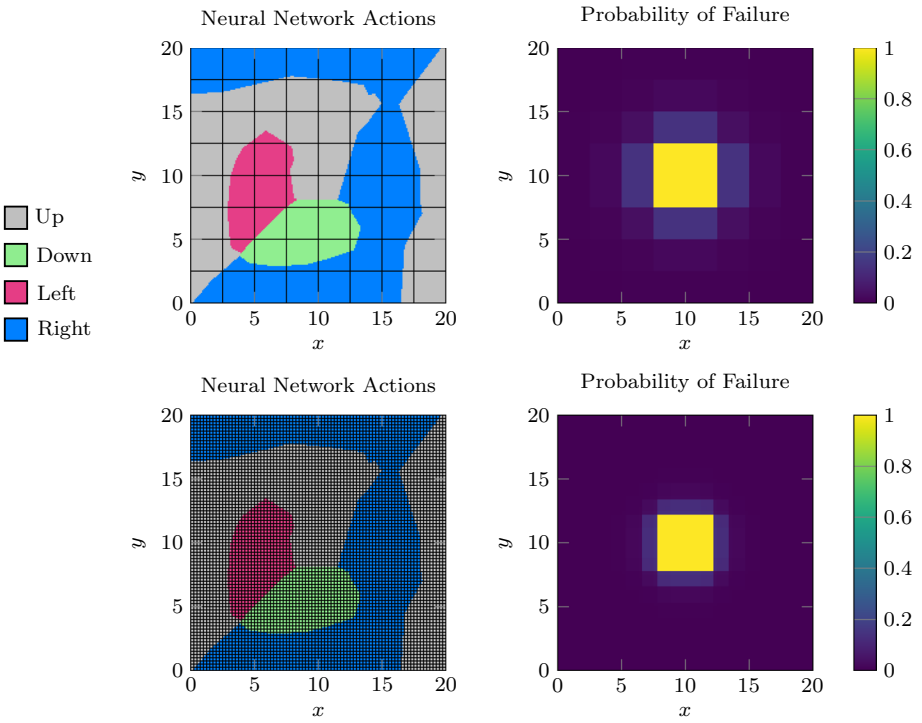

Fig. 3 Overapproximated probability of falling into the pit for different cell partitions. The plots in the left column show the cells plotted on top of the neural network policy, while the plots in the right column show the overapproximated probability of falling into the pit from each cell

4.1 Offline reduction: adaptive verification

As described in Sect. 3.1, overapproximation error grows with the number of possible actions in a cell, and cells with only one possible action will have no overapproximation error in the policy. Therefore, in order to partition our space in a way that minimizes policy overapproximation, we seek to minimize the volume of the input space occupied by cells that have multiple possible actions. Figure 4 shows an example of two possible partitions of the input space for an example policy. While both partitions contain the same number of cells, the second partition has a smaller area of the input space covered by cells with multiple possible actions and therefore a smaller overapproximation error. Our goal is to develop a verification strategy that will automatically generate a partition similar to the rightmost partition in Fig. 4.

We use an adaptive verification strategy summarized in algorithm 1 to obtain $\tilde{\pi}$. We begin with a single cell that encompasses all of $\mathcal{S}$. Each time we evaluate a cell, we run the verification tool to check which actions are possible in the specified cell to obtain $\mathcal{A}_c$. If $\mathcal{A}_c$ contains more than one action and the cell exceeds the minimum cell size, we split the cell into smaller cells. Splitting continues until all cells are either below the minimum cell width in the splitting dimension or have a single action in $\mathcal{A}_c$.

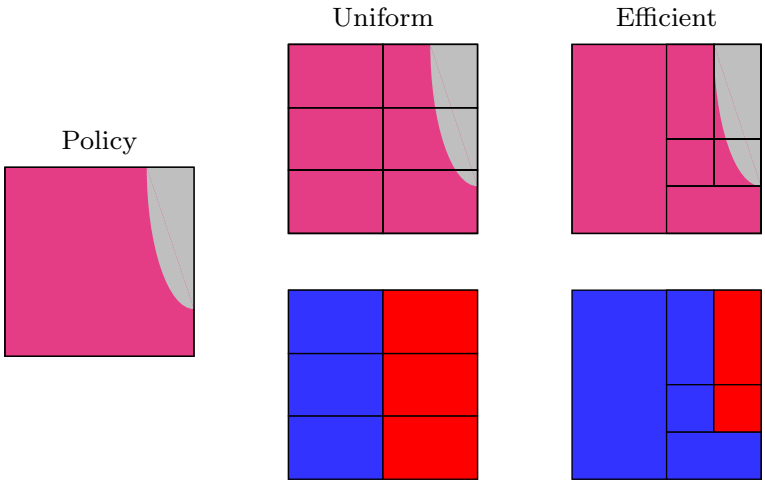

Fig. 4 Two possible partitions of the input space for the simple policy with two possible actions shown on the left. The pink region corresponds to the first action, and the gray region corresponds to the second action. The top row shows an overlay of the partition on the policy, while bottom row shows the corresponding number of actions in each cell. Blue cells have one possible action, while red cells have two possible actions (Color figure online)

Algorithm 1 Adaptive Verification

```
1: function AdaptiveVerification(neuralNetwork, inputCell, minCellSize)
2:     initialize s to empty stack
3:     push inputCell onto s
4:     while s is not empty
5:         c ← pop(s)
6:         A_c ← VerificationTool(neuralNetwork, c)
7:         if length of A_c > 1 and size of c > minCellSize
8:             split c according to splitting strategy
9:             push the resulting cells to s
10:    return π̃
```

Because calls to neural verification tools are computationally expensive, we want to select a splitting strategy that will allow us to minimize the number of calls. We tested the following two splitting strategies.

- *All split* splits the cell along all dimensions at the midpoint
- *Informed split* attempts to speed up the verification process by first evaluating the neural network at the corners of the cell. If the corner points evaluate to different actions, we know that the verification tool would return multiple possible actions. Therefore, we can split the cell without calling it. Furthermore, we can use the evaluations of the corner points to select the dimensions to split.

Figure 5 demonstrates the informed split strategy for different corner evaluations. If the adjacent actions are the same across a particular dimension, we do not split along that

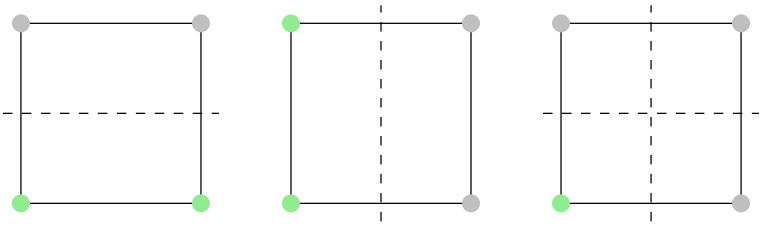

Fig. 5 Example splitting of three cells for the informed splitting strategy based on the actions at the corners. The colored dots represent the actions at the corners with different colors indicating different actions

dimension. For example, the adjacent actions in the first dimension in the leftmost cell of Fig. 5 are equal, so we do not split in the first dimension.

The adaptive verification algorithm encourages small cells near the decision boundaries of the neural network policy, and larger cells in continuous regions of the same action. This result is illustrated by Fig. 6, which shows the result of applying algorithm 1 to the continuum world example with each splitting strategy. Both splitting strategies focus on the decision boundaries of the network. By only splitting along particular dimensions, the informed split strategy results in fewer cells overall.

4.2 Online reduction: state abstraction

While offline error reduction addresses policy overapproximation error, it does not address the second source of error. In order to reduce both types of error, we add online error reduction techniques to the model checking process. The online overapproximation error reduction techniques presented in this work are inspired by state abstraction techniques developed to solve for the value function of MDPs with large state spaces (Munos and Moore 2002). State abstraction relies on the assumption that large portions of the state space will have low variability in the policy or value function, while other more critical regions of the state space will require a finer resolution for accuracy. Some portions of the state space are safety-critical; however, a large portion of the state space will have a low probability estimate regardless of the action taken. During the solving process, we use heuristics based on our current probability estimate to determine critical portions of the state space. Splitting

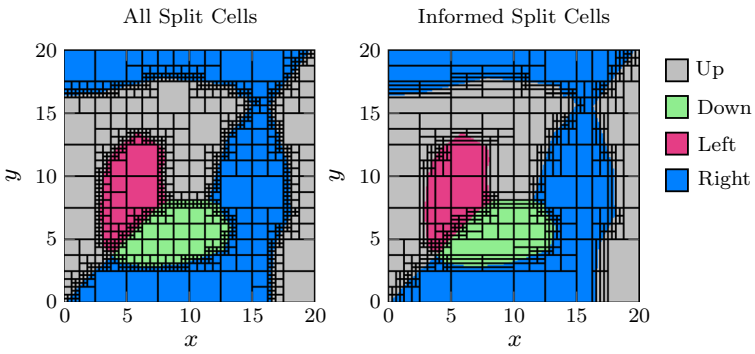

Fig. 6 Resulting cell partition when applying algorithm 1 to the continuum world explanatory example plotted on top of the neural network policy for each splitting strategy

cells in these critical regions allows us to significantly reduce overapproximation error during the solving process without making cells in the input space unnecessarily small.

We address the worst-case transition error with an online splitting heuristic based on the maximum range of probability values for next cells. This range is computed as

$$\text{transitionRange} = \max_i \left[\max_{c' \in \mathcal{C}'_i} \Pr^{\tilde{\pi}}(c') - \min_{c' \in \mathcal{C}'_i} \Pr^{\tilde{\pi}}(c') \right] \tag{7}$$

As an example, the worst-case transition range for the example shown in Fig. 2 is calculated as

$$\text{transitionRange} = \max(0.0 - 0.0, 0.13 - 0.03, 0.13 - 0.12, 0.02 - 0.0) = 0.1 \tag{8}$$

If this range is high, the worst-case transition overapproximation error is likely to be high. Therefore, we split the cell if this range exceeds a specified threshold and the cell is larger than a minimum cell size. The threshold can be tuned by the user to achieve a desired resolution. Splitting the cell will shrink the region of next states for each new cell and reduce the spread of overapproximation error.

The second online splitting heuristic aims to reduce policy overapproximation error and applies to cells that have multiple actions in $\mathcal{A}_c$. If a cell has multiple actions, we compute the range of the probabilities when taking each action as

$$\text{actionRange} = \max_{a \in \mathcal{A}_c} \Pr^{\tilde{\pi}}(c, a) - \min_{a \in \mathcal{A}_c} \Pr^{\tilde{\pi}}(c, a) \tag{9}$$

If this range exceeds a threshold and the cell is larger than the minimum cell size, we split the cell and rerun the verification on the resulting smaller cells.

4.3 Algorithm summary

The neural network model checking methods established in this work are summarized in algorithm 2. Inputs to the algorithm include the neural networks to verify, the reachable set encoding the property we wish to verify, an initial cell that covers the entire network input space, the minimum cell size, and the transition and action thresholds. The last three input parameters can be tuned to achieve desired accuracy. A smaller minimum cell size and lower values for the thresholds will result in smaller overapproximation error and therefore a more accurate estimate of the probabilities.

Algorithm 2 Neural Network Model Checking

function CHECK(neuralNetwork, $\mathcal{B}$, inputCell, minCellSize, transitionThreshold, actionThreshold)
  $\tilde{\pi} \leftarrow$ ADAPTIVEVERIFICATION(neuralNetwork, inputCell, minCellSize)
  $\Pr^{\tilde{\pi}}(c) \leftarrow 0$ for all $c \notin \mathcal{B}$
  $\Pr^{\tilde{\pi}}(c) \leftarrow 1$ for all $c \in \mathcal{B}$
  repeat
    initialize s to an empty stack
    push c onto s for all $c \in \mathcal{C}$
    while s is not empty
      $c \leftarrow \text{pop}(s)$
      ranges $\leftarrow \emptyset$
      for $a \in \mathcal{A}_c$
        compute $\mathcal{C}'_{1:n}$ from c for action a
        add $\max_i \left[\max_{c' \in \mathcal{C}'_i} \Pr^{\tilde{\pi}}(c') - \min_{c' \in \mathcal{C}'_i} \Pr^{\tilde{\pi}}(c')\right]$ to ranges
      transitionRange $\leftarrow$ maximum of ranges
      if transitionRange $>$ transitionThreshold and size of $c >$ minCellSize
        split c
        add resulting cells to s
      else
        $\Pr^{\tilde{\pi}}(c, a) \leftarrow \sum_{c' \in \mathcal{C}} T(c' \mid c, a) \max_{a' \in \mathcal{A}_c} \Pr^{\tilde{\pi}}(c', a')$ for all $a \in \mathcal{A}_c$
        actionRange $\leftarrow \max_{a \in \mathcal{A}_c} \Pr^{\tilde{\pi}}(c, a) - \min_{a \in \mathcal{A}_c} \Pr^{\tilde{\pi}}(c, a)$
        if actionRange $>$ actionThreshold and size of $c >$ minCellSize
          split c
          add resulting cells to s
  until convergence
  return $\Pr^{\tilde{\pi}}$

The first step in the algorithm is to obtain an overapproximation of the neural network policy using the adaptive verification method in algorithm 1. Next, the probability is initialized to zero for all cells except those that overlap with the reachable set $\mathcal{B}$, which are initialized to a probability of one. After these preprocessing steps, we begin value iteration with our online splitting heuristics. For each cell, we first compute the transition range according to Eq. (7). If the cell satisfies the worst-case transition splitting criterion, we split the cell. Otherwise, we perform the Bellman update (Eq. 4) to compute the probability of collision for taking each action in $\mathcal{A}_c$ from the cell. Using these probabilities and Eq. (9), we can compute the action range and decide once again whether to split the cell.

Figure 7 compares the overapproximated probability of falling into the pit when taking two different approaches to partitioning the state space into cells. The first approach represents the naïve approach in which the space is uniformly partitioned into small cells. The second approach follows our algorithm, applying adaptive verification with the informed split strategy and using the online overapproximation error reduction methods during the solving process. The partition using the second approach results in small cells in regions of the state space near the pit and the decision boundaries of the network. Using our overapproximation error reduction methods, we are able to obtain similar probability estimates with significantly fewer cells in the partition.

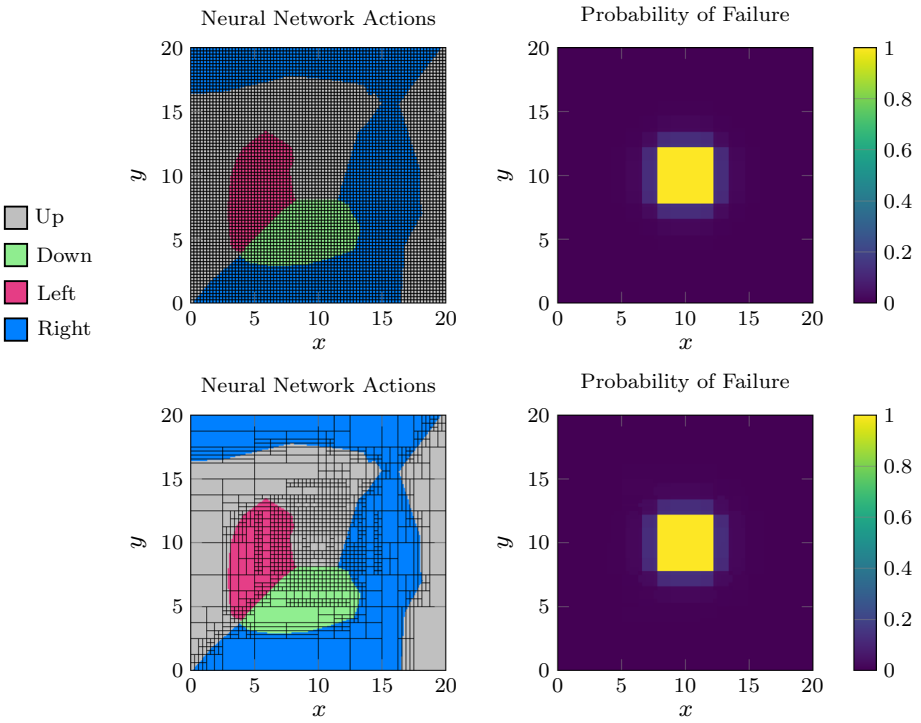

Fig. 7 Overapproximated probability of falling into the pit using a naïve approach to partitioning the state space into cells (top) compared to using adaptive verification with the informed splitting strategy along with the online heuristics. The plots in the left column show the cells plotted on top of the neural network policy, while the plots in the right column show the overapproximated probability of falling into the pit from each cell

4.4 Complexity

The overapproximation error reduction methods presented in Sects 4.1 and 4.2 are key contributors to the overall complexity of the algorithm. In the worst case, algorithm 1 will result in a uniform splitting of the input space in which all cells have the minimum cell size. Therefore, the worst-case complexity of algorithm 1 is exponential in the input dimension of the neural network controller. Despite this worst-case complexity, control policies in practice tend to be structured such that large, continuous regions of the state space correspond to the same action. Our adaptive verification approach takes advantage of this structure by allowing large regions of the same action to be grouped into a single cell while only decreasing cells on the decision boundaries of the network to the minimum cell size.

The complexity of algorithm 1 observed in practice depends on the splitting strategy. Because the all split strategy simply splits along every dimension, the number of new cells created on each split is always exponential in the input dimension. In contrast, while still exponential in the worst case, the informed split strategy limits the number of new cells created in practice by selecting a subset of dimensions to split using the evaluations at the corners. We note that the number of corner points of a cell grows exponentially with the input dimension, so this strategy will be intractable for controllers with high-dimensional

inputs. In this case, one could consider using the gradient-based splitting heuristics used in prior work on neural network verification to select a subset of dimensions to split (Wang et al. 2018).

The online reduction strategies described in Sect. 4.2 also have worst-case exponential complexity in the input dimension. However, similarly to algorithm 1, the strategies exploit the structure of the problem to limit this complexity in practice by only splitting cells in safety-critical regions. Another contributor to algorithm complexity is calls to the neural network verification tool. Depending on the verification algorithm, queries on larger cells tend to have greater complexity. Therefore, there is a tradeoff between using large cells to limit the total number of cells in the partition and keeping the cells small enough to be effectively handed by a neural network verification tool. Furthermore, the size and type of neural networks we can verify with our algorithm is limited by the current capabilities of neural network verification tools. These tools tend to perform best on small neural networks with simple architectures and rectified linear unit (ReLU) activations (Liu et al. 2021).

As a result of the worst-case complexity in the input dimension, our algorithm is most effective on neural network controllers with relatively low-dimensional inputs. Controllers that take the physical state of the system as input tend to have this property; however, our method will not scale well to controllers with high-dimensional inputs such as images.

5 Application: collision avoidance neural networks

We use the aircraft collision avoidance problem as an example application for our methods. Aircraft collision avoidance provides a compelling, real-world example of a safety-critical application in which neural network controllers provide a substantial benefit and has been used as a benchmark in previous work on neural network verification. Specifically, neural networks have been demonstrated as space-efficient controllers for a family of aircraft collision avoidance systems called the Airborne Collision Avoidance System X (ACAS X) (Julian et al. 2016, 2019a). ACAS X relies on a large numeric lookup table to provide optimized advisories during flight (Kochenderfer and Chryssanthacopoulos 2011; Kochenderfer et al. 2012; Olson 2015; Owen et al. 2019). Julian et al. (2019a) showed that it is possible to decrease the memory footprint of the table by training a neural network to take its place. The neural network representation decreases the required storage by a factor of 1,000 while maintaining comparable performance to the table in simulation.

The collision avoidance neural networks used in this work are based on the networks in the VerticalCAS repository developed by Julian and Kochenderfer (2019b). The repository contains an open source collision avoidance logic loosely modeled after the vertical logic used in ACAS X (Julian and Kochenderfer 2019b). The logic is designed to prevent near mid-air collisions (NMACs), which are defined as a simultaneous loss of separation to less than 500 ft horizontally and 100 ft vertically. To apply model checking to the problem, we must first formulate it as a Markov decision process. We use a similar formulation to what was used to generate the lookup table for ACAS X.

State Space The state space for VerticalCAS consists of five state variables. Table 1 summarizes the variables and their ranges, and Fig. 8 provides a visual representation of the state variables. The first three variables summarize the relative positioning and vertical rate of the ownship and intruder aircraft. The next state variable, τ, compactly summarizes the horizontal geometry by representing the time until the horizontal separation between the two aircraft is

Table 1 VerticalCAS state variables

Variable	Description	Units	Range (low:high)
h	Relative altitude of intruder	ft	− 8000:8000
$\dot{h}_0$	Ownship vertical rate	ft/s	− 100:100
$\dot{h}_1$	Intruder vertical rate	ft/s	− 100:100
τ	Time to loss of lateral separation	s	0:40
a_{prev}	Previous advisory	N/A	N/A

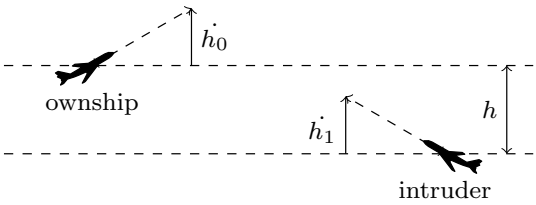

Fig. 8 Visual representation of verticalCAS state variables

Table 2 VerticalCAS action space

Action	Description
COC	Clear of conflict
DNC	Do not climb
DND	Do not descend
DES1500	Descend ≥ 1500 ft/min
CL1500	Climb ≥ 1500 ft/min
SDES1500	Strengthen descent to ≥ 1500 ft/min
SCL1500	Strengthen climb to ≥ 1500 ft/min
SDES2500	Strengthen descent to ≥ 2500 ft/min
SCL2500	Strengthen climb to ≥ 2500 ft/min

less than 500 ft. Finally, adding the previous advisory to the state space allows us to penalize a reversal or strengthening of an advisory while still satisfying the Markov property.

Action Space The action space consists of the advisories that the collision avoidance system will provide to the aircraft during flight. The logic has nine possible advisories, which are summarized in Table 2. All advisories except COC represent an alert and command the aircraft to a particular vertical rate range. The COC advisory indicates that there is currently no threat of collision with an intruding aircraft.

Transition Model The transition model uses the following linear dynamics model

$$\begin{aligned} h &\leftarrow h + \dot{h}_1 + 0.5\ddot{h}_1 - \dot{h}_0 - 0.5\ddot{h}_0 \\ \dot{h}_0 &\leftarrow \dot{h}_0 + \ddot{h}_0 \\ \dot{h}_1 &\leftarrow \dot{h}_1 + \ddot{h}_1 \\ \tau &\leftarrow \tau - 1 \\ a_{\text{prev}} &\leftarrow a \end{aligned} \tag{10}$$

We assume a one second time step corresponding to the 1 Hz update frequency of the collision avoidance system. The dynamics model is made stochastic by assuming distributions over the accelerations of the ownship and intruder. The intruder is assumed to have an equal chance of following accelerations $-g/8$, $g/8$, and 0 ft/s^2 as in Julian and Kochenderfer (2019a). The ownship follows accelerations $\ddot{h}_{1-3}$ based on its previous advisory with associated probabilities $p_{1:3}$ shown in Table 3. This transition model contains two features that add robustness. First, the ownship is assumed to follow the accelerations associated with its previous advisory (rather than its current advisory), which represents a short delay in the aircraft's response to the advisory. Additionally, the ownship is assumed to accelerate in the opposite direction of its advisory 20% of the time to further incorporate errors in aircraft response. For example, if a human pilot is responsible for executing the collision avoidance maneuver, they may not respond instantaneously to an advisory and could be accelerating in a direction opposite the advisory. The probabilistic safety guarantees presented in this paper are based on this stochastic dynamics model.

The reward model balances between safety and efficiency with a high penalty for an NMAC and relatively smaller penalties for alerting advisories. Using this formulation, we can solve for the optimal policy using value iteration. Traditionally, the final action-values result in a large numeric lookup table. To reduce the on-board memory requirements, Julian et al. (2019a) train a neural network representation to approximate the action-value function. One network is trained for each discrete previous advisory. The values of the other four state variables in Table 1 make up the four-dimensional input to the network, and the approximate value of each action makes up the nine-dimensional output. Each network has five hidden layers with 25 units each that use rectified linear unit (ReLU) activation functions. The networks used in this work can be found at https://github.com/sisl/AdaptiveVerification/.

Figure 9 shows a comparison of the neural network policy and lookup table policy for a slice of the state space. The neural network policy closely approximates the table policy with a few subtle differences that are visible in the plot. Even though the alerting regions in the neural network policy appear continuous, the plot is generated by evaluating a finite number of points in the state space and does not guarantee this property. For example, while the region highlighted on the neural network policy in Fig. 9 appears to evaluate to CL1500 at all points within it, we cannot guarantee this property just by examining the plotted points. To provide a guarantee that all advisories in that region are in fact CL1500, we use a neural verification tool to obtain $\mathcal{A}_c$. For this application, we use the Reluval

Table 3 Transition accelerations

Previous action	Probabilities (p_{1-3})	Accelerations ($\ddot{h}_{1-3}$)
COC	[0.34, 0.33, 0.33]	[0.0, -g/3, g/3]
DNC	[0.50, 0.30, 0.20]	[-g/3, -g/2, g/3]
DND	[0.50, 0.30, 0.20]	[g/3, g/2, -g/3]
DES1500	[0.50, 0.30, 0.20]	[-g/3, -g/2, g/3]
CL1500	[0.50, 0.30, 0.20]	[g/3, g/2, -g/3]
SDES1500	[0.50, 0.30, 0.20]	[-g/2.5, -g/2, g/3]
SCL1500	[0.50, 0.30, 0.20]	[g/2.5, g/2, -g/3]
SDES2500	[0.50, 0.30, 0.20]	[-g/2.5, -g/2, g/3]
SCL2500	[0.50, 0.30, 0.20]	[g/2.5, g/2, -g/3]

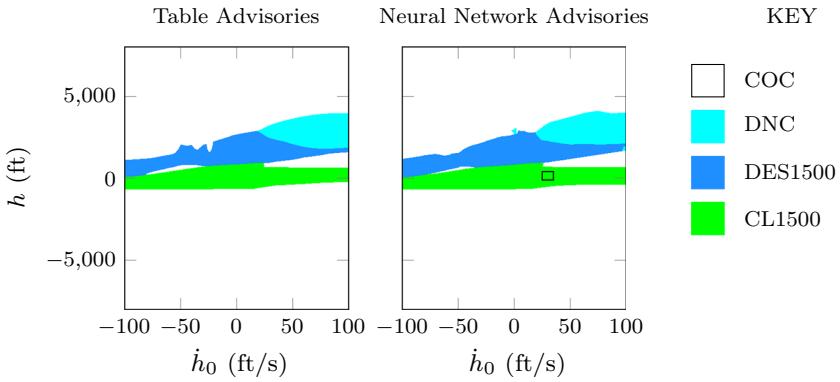

Fig. 9 Comparison of table policy to neural network policy for a slice of the state space. The intruder vertical rate is fixed at −90 ft/s, τ is fixed at 5 s, and the previous advisory is COC

algorithm to verify each cell due to its fast performance on the collision avoidance networks (Liu et al. 2021; Wang et al. 2018).

We can craft an LTL formula to determine the probability of an NMAC by using the temporal operator "eventually," written as F. Let the atomic proposition N represent whether or not a cell belongs to the NMAC region ($\tau = 0$ s and $h < 100$ ft). The LTL formula of interest is FN, read as "eventually N." The probability of satisfying this formula corresponds to the probability that a state will eventually reach an NMAC state. This problem is easily converted to a reachability problem. Let $\mathcal{B}$ represent the collection of all cells that satisfy N. We seek to find the maximum probability of reaching cells in $\mathcal{B}$ while following the overapproximated neural network policy.

6 Results

For ease of computation and visualization, we tested our methods on a two-dimensional version of the collision avoidance problem in which the intruder vertical rate is fixed at −90 ft/s before testing on the full scale model. Since most of the overapproximation error in our initial experiments seemed to be concentrated at high vertical rates, we selected −90 ft/s for the intruder vertical rate to understand how our method performs in a challenging area of the state space. All other aspects of the problem, including the neural networks used for verification, remain the same. By taking this approach, we were able to better understand the effects of each aspect of the algorithm. Therefore, the results summarizing the effects of the overapproximation reduction techniques were generated using the two-dimensional model. After providing intuition with these results, we present the results of the full scale model.

6.1 Adaptive verification

We tested both the all split and informed split adaptive verification splitting strategies to obtain the overapproximated policy $\tilde{\pi}$. Figure 10 shows the results of both splitting strategies when the intruder vertical rate is fixed at −90 ft/s, and Table 4 shows a comparison of runtime for both the two-dimensional and full scale model for each strategy. We also

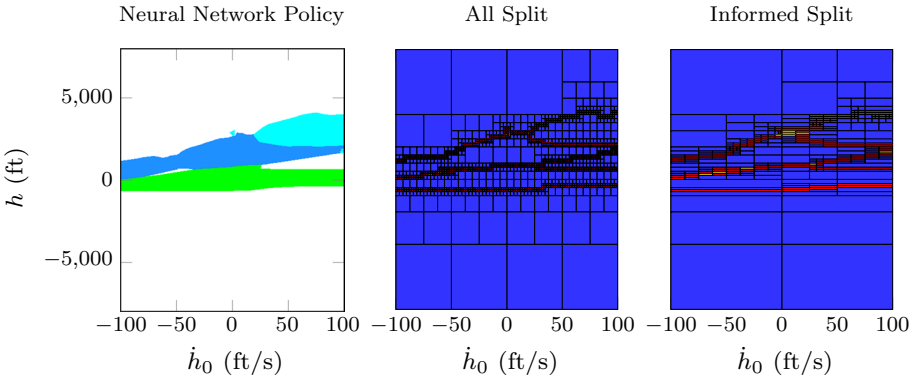

Fig. 10 Comparison of the resulting state space partition for the adaptive splitting strategies. The intruder vertical rate is fixed at −90 ft/s, τ is 5 s, and the previous advisory is COC. Cells that are colored blue have only one possible advisory in $\mathcal{A}_c$, red cells have two possible advisories, and yellow cells have three or more possible advisories (Color figure online)

Table 4 Adaptive verification timing results

Splitting strategy	$\dot{h}_1$ Fixed time (s)	Full scale time (s)
Uniform	235.5	16,000 (est.)
All split	21.9	2500.3
Informed split	5.5	655.6

include the runtime of a uniform, non-adaptive splitting strategy in which all cells are the minimum cell size used in the adaptive strategies. For the full-scale model, the non-adaptive runtime was estimated based on the time required to verify a single cell of the minimum cell size. Time trials were run on a single 4.20 GHz Intel Core i7 processor. Both adaptive splitting strategies result in small cells around the decision boundaries of the network; however, the informed splitting strategy results in fewer cells.

The adaptive strategies require fewer calls to the verification tool and therefore perform faster than a non-adaptive strategy. Additionally, whenever a cell is split and the resulting cells are reverified, the verification tool only needs to check for the actions that were possible in the larger cell. Informed split is faster than all split because it makes even fewer Reluval calls. Checking the corners to inform the split prevents unnecessary calls and results in fewer cells in the final partition. Due to its speed, the informed split strategy was selected for the rest of the analysis in this work.

While the adaptive verification technique presented here addresses policy overapproximation error for model checking, it may also be used on its own to analyze neural network policies and detect decision boundaries. The algorithm is an anytime algorithm, and the resolution can be controlled using the minimum cell size parameter. Figure 11 shows the results of running adaptive verification as the parameter is decreased. To minimize overapproximation error, we want to maximize the amount of the state space covered by cells with a single advisory in them (blue cells). Figure 11 demonstrates that as the minimum cell size decreases, overapproximation error in the policy also decreases.

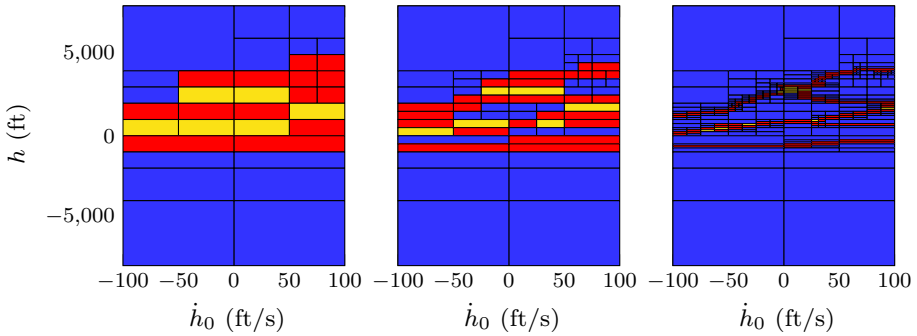

Fig. 11 Adaptive verification results using informed split for decreasing minimum cell sizes (left to right). The intruder vertical rate is fixed at −90 ft/s, τ is 5 s, and the previous advisory is COC. Cells that are colored blue have only one possible advisory in $\mathcal{A}_c$, red cells have two possible advisories, and yellow cells have three or more possible advisories (Color figure online)

After obtaining the overapproximated network policy using adaptive verification, we can run model checking on the cells. Figure 12 shows the results for a slice of the state space. Assuming that an encounter gets initiated with 40 s until loss of horizontal separation, a safe policy should have a low probability of NMAC for all states at $\tau = 40$ s; however, the probability values at $\tau = 40$ s after model checking are close to one, and we therefore cannot provide a safety guarantee with this adaptive verification method alone. We need to introduce the online error reduction techniques.

6.2 Online reduction

Figure 13 displays the model checking results with the online worst-case transition splitting heuristic for the same slice of the state space shown in Fig. 12.

The algorithm splits cells in a densely packed band along the safety-critical region of the state space where a collision is imminent. The intruder is rapidly descending at 90 ft/s,

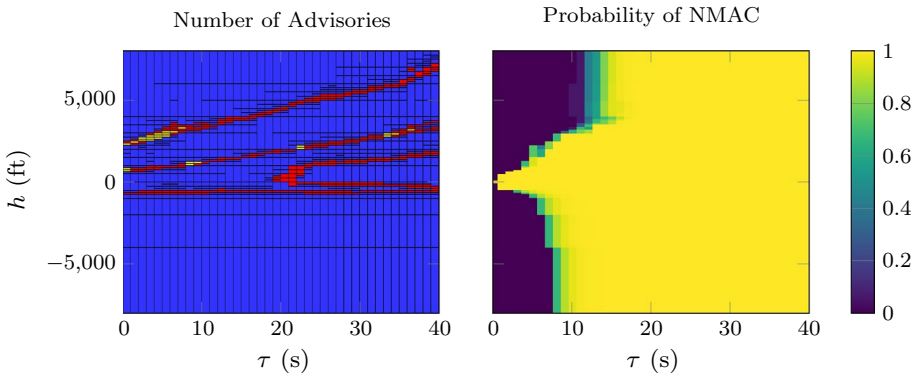

Fig. 12 State space partition (same color scheme as Fig. 10) and overapproximated probability of NMAC with no online overapproximation reduction. The intruder vertical rate is fixed at −90 ft/s, the ownship vertical rate is 0 ft/s, and the previous advisory is COC

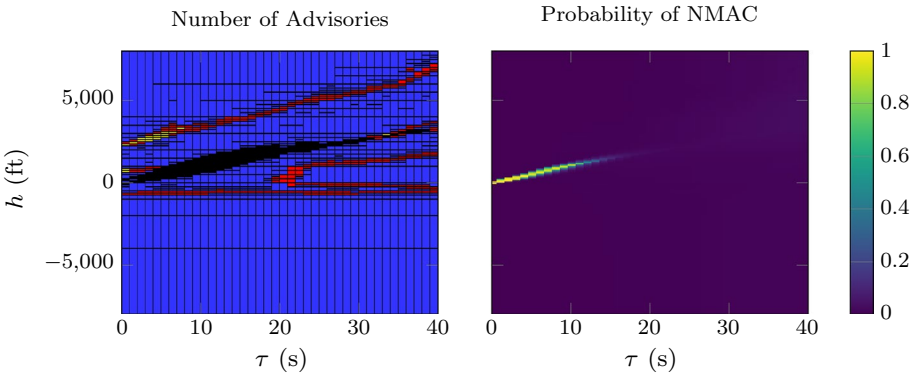

Fig. 13 State space partition (same color scheme as Fig. 10) and overapproximated probability of NMAC with the worst-case transition splitting heuristic. The intruder vertical rate is fixed at −90 ft/s, the ownship vertical rate is 0 ft/s, and the previous advisory is COC

so the region directly above the ownship is most dangerous. Outside of this band, the state space partition remains untouched, indicating that the online splitting heuristic only splits states that require a finer resolution.

The overapproximation error in Fig. 13 is much lower than the error in Fig. 12, and we can now obtain meaningful information from the estimated probabilities. For example, the probabilities are highest at low values of τ when the intruder is above the ownship. The high intruder descent rate results in the upward sloping band of high probability extending away from the NMAC region at $\tau = 0$. At $\tau = 40$ s, the probabilities of collision are significantly lower with a maximum probability of 0.0305 among all cells. The splitting threshold can be tuned to achieve a desired resolution as shown in Fig. 14. The highest threshold only splits cells in the extremely safety-critical region near $\tau = 0$. As the threshold is decreased, the band of tightly packed cells extends further away from $\tau = 0$ and overapproximation error decreases.

Figure 15 shows the results when the policy overapproximation heuristic is used in addition to the worst-case transition heuristic. Beyond the splits due to the worst-case transition heuristic, this heuristic results in splits in overapproximated regions at higher values of τ. There is no significant difference in the visualization of probabilities between Figs. 13 and 15, and the maximum probability of NMAC at $\tau = 40$ s drops from 0.0305 to 0.0268. The addition of this splitting heuristic did not add a significant benefit over the worst-case transition heuristic, possibly due to the fact that adaptive verification already addresses policy overapproximation error.

Table 5 summarizes the effect of each overapproximation error reduction technique on the maximum overapproximated probability of NMAC for cells at $\tau = 40$ s. We also provide the number of cells in the final partitioning and the time required to perform the model checking to illustrate the added complexity of each technique. All time trials for this experiment were performed on 5 cores of a 4.20 GHz Intel Core i7 processor.

The adaptive verification technique alone is not enough to reduce overapproximation error to obtain a meaningful estimate of the probability of NMAC. Adaptive verification addresses only policy overapproximation error but does nothing to reduce worst-case transition error. When we add the online splitting heuristic to address worst-case transition error and decrease the splitting threshold, the probability estimate decreases to a more

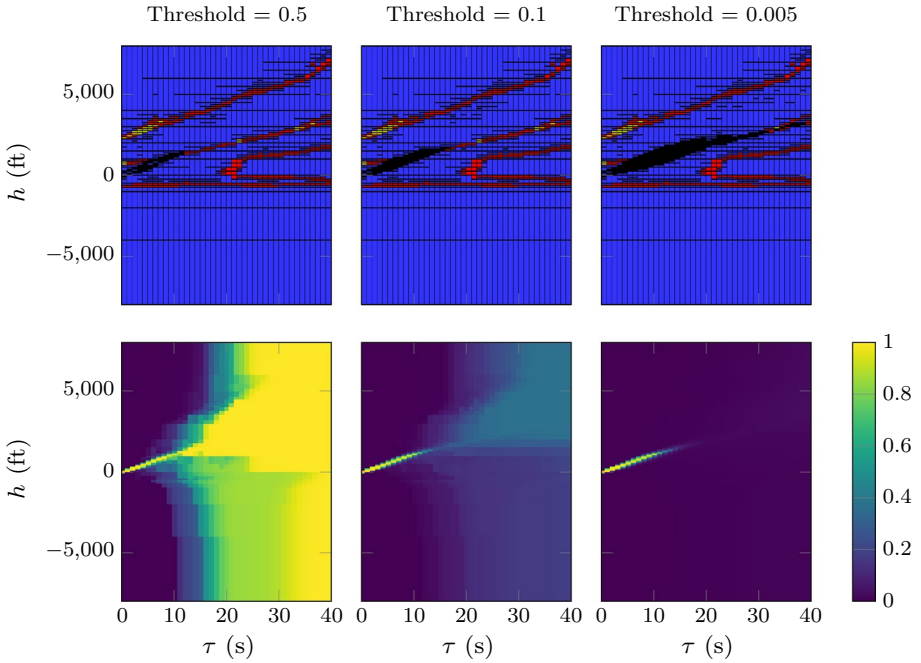

Fig. 14 State space partition (same color scheme as Fig. 10) and overapproximated probability of NMAC using various thresholds for the worst-case transition splitting heuristic. The intruder vertical rate is fixed at −90 ft/s, the ownship vertical rate is 0 ft/s, and the previous advisory is COC

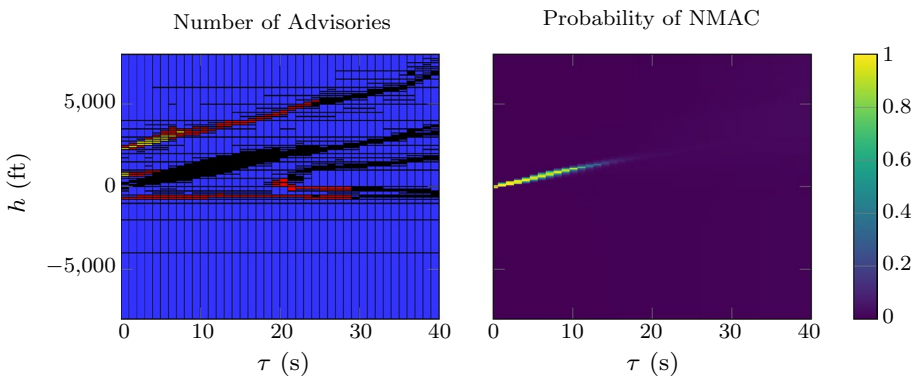

Fig. 15 State space partition (same color scheme as Fig. 10) and overapproximated probability of NMAC with both the worst-case transition splitting heuristic and the policy overapproximation splitting heuristic. The intruder vertical rate is fixed at −90 ft/s, the ownship vertical rate is 0 ft/s, and the previous advisory is COC

meaningful result. We can guarantee that the probability of NMAC is less than 0.0305. Adding the online splitting heuristic to further reduce policy overapproximation in key areas of the state space does not have as much of an effect on the resulting probability. Nevertheless, it changes the guarantee to a 0.0268 probability of NMAC.

Table 5 Effect of error reduction methods on overapproximated probability of NMAC

Error Reduction	Transition Threshold	Action Threshold	Cells	Time (s)	Probability
Adaptive verification	–	–	3.4×10^5	157	1.0
Worst-case transition	0.5	–	6.9×10^5	218	1.0
Worst-case transition	0.1	–	2.3×10^6	565	0.372
Worst-case transition	0.005	–	8.9×10^6	1546	0.0305
All techniques	0.005	0.005	1.2×10^7	6336	0.0268

While the overapproximation error in the probability estimate decreases when we apply the online splitting heuristics, the number of cells in the final partition and time required to perform the model checking increases. Table 5 therefore shows a tradeoff between the error in the final estimate and the overall complexity of the algorithm. However, the online splitting heuristics still allow us to exploit the structure of the problem to ensure that splits only occur in critical parts of the state space. If we were to instead naïvely partition the state space uniformly into cells of the minimum cell size, our final partition would have 2.36×10^8 cells. Comparing to the results in Table 5, even the partition that uses all overapproximation error reduction techniques has less than 5% of the cells required for a uniform partition.

6.3 Full scale model

After gaining intuition using the two-dimensional model, we applied our method to determine the maximum probability of collision on the full scale model. For computational reasons, the action range threshold was slowly increased throughout the solving process. To analyze the quality of our probability estimates, we generate two baseline probability comparisons. The first comparison is the estimated probability of NMAC when using the large numeric lookup table that the neural network is meant to approximate. This probability is calculated by performing traditional MDP model checking on the table policy using the method outlined in Sect. 2.2 with the same model used to evaluate the neural network. We use multilinear interpolation to determine probabilities at points in the state space that do not correspond directly to table entries, so the result is not guaranteed to be an overapproximation. We also compare with the probability of NMAC detected through 1,000 Monte Carlo simulations from various points in the state space. While Monte Carlo simulations cannot provide any formal guarantees, they provide a good approximation to the probability of NMAC.

The results for one slice of the state space are shown in Fig. 16. All three plots show similar trends. The model checking outputs for the table and neural network are similar with slightly higher probabilities for the neural network. The probabilities are also similar to the Monte Carlo probabilities with the same cells showing high probability of collision; however, the region of high probability is slightly larger in the model checking results. It is clear that the model checking probabilities represent an overapproximation.

Figure 17 shows the maximum probability of collision among all cells as τ is increased for the same slice of the state space shown in Fig. 16. At $\tau = 0$ s, the maximum probability of NMAC is one, corresponding to cells in the NMAC region. As τ

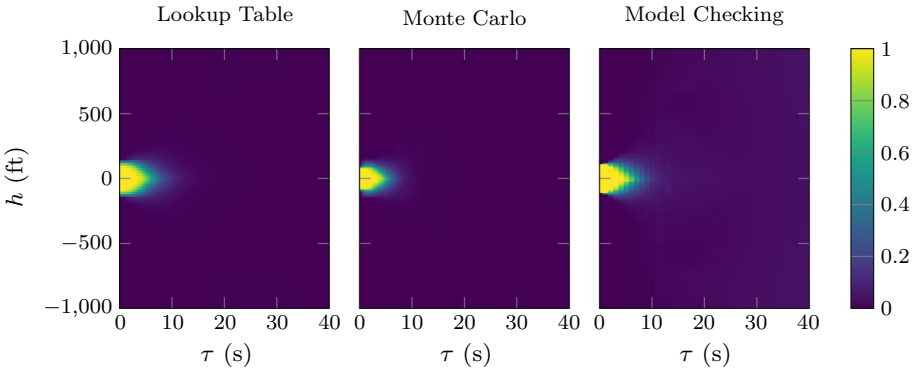

Fig. 16 Comparison of lookup table probabilities of NMAC estimated using traditional MDP model checking, neural network probabilities of NMAC estimated using Monte Carlo simulations, and overapproximated neural network probabilities of NMAC using the model checking formulation in this work. The ownship and intruder vertical rates are both 0 ft/s, and the previous advisory is COC

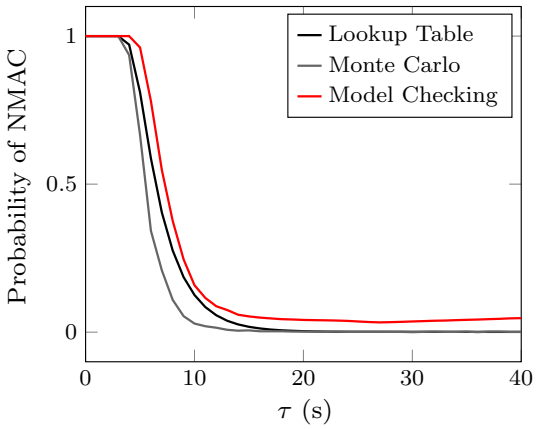

Fig. 17 Probability of NMAC when previous advisory is COC as time to loss of separation increases. The ownship and intruder vertical rates are fixed at 0 ft/s

increases for each method, there is a sharp dropoff in probability of NMAC. The overapproximated model checking probabilities are greater than or equal to both the lookup table and Monte Carlo probabilities at all points. The curve, however, remains close to the other curves and represents a tight overapproximation. Furthermore, unlike the Monte Carlo estimate and lookup table model checking estimate, the neural network model checking estimate represents a guarantee on the performance of the neural network policy. The final model checking probability is 0.0473 at $\tau = 40$ s. Therefore, with respect to the stochastic dynamics model in Sect. 5, we can guarantee that the probability of NMAC is less than 0.0473 when the intruder and ownship are in level flight.

We also examined the probability of NMAC when the aircraft are climbing or descending. Figure 18 contains the results when the ownship is climbing at 60 ft/s in comparison to the table and Monte Carlo results, and Fig. 19 shows the vertical rates when both aircraft are climbing at a rate of 60 ft/s. The trends in the probabilities match the trends seen in the lookup table and Monte Carlo results. In Fig. 18, the region of high probability of NMAC extends above the ownship because it starts in a climb.

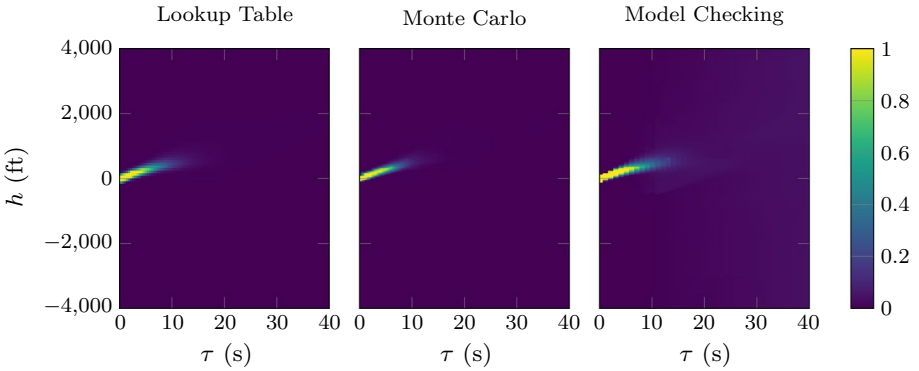

Fig. 18 Comparison of lookup table probabilities of NMAC estimated using traditional MDP model checking, neural network probabilities of NMAC estimated using Monte Carlo simulations, and overapproximated neural network probabilities of NMAC using the model checking formulation in this work. The ownship vertical rate is 60 ft/s, the intruder vertical rate is 0 ft/s, and the previous advisory is COC

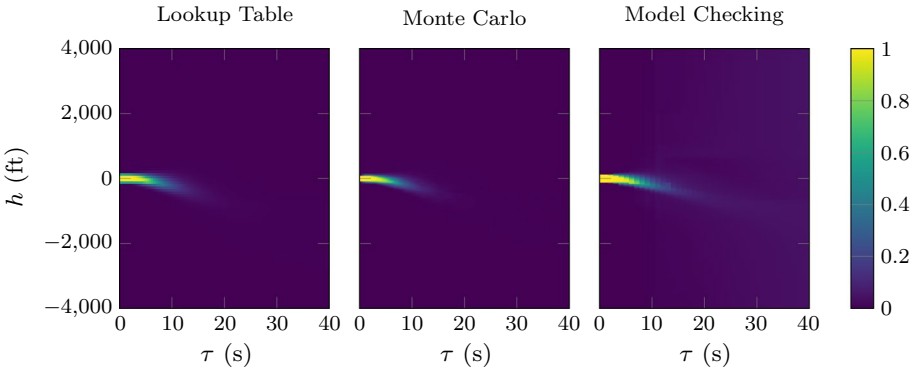

Fig. 19 Comparison of lookup table probabilities of NMAC estimated using traditional MDP model checking, neural network probabilities of NMAC estimated using Monte Carlo simulations, and overapproximated neural network probabilities of NMAC using the model checking formulation in this work. The ownship and intruder vertical rates are both 60 ft/s, and the previous advisory is COC

When both aircraft are climbing at the same rate in Fig. 19, the two aircraft are on a direct collision path when they are co-altitude. For this reason, probabilities are high in this region when τ is small. As the time to collision increases, the region shifts downward slightly since the intruder is not executing collision avoidance maneuvers and is therefore most dangerous at altitudes below the ownship. The overapproximated probabilities are again higher than the Monte Carlo probabilities, but the maximum probability of NMAC with these starting states decreases slightly to 0.0462 and 0.0437, respectively.

The maximum probability of NMAC among all states at $\tau = 40$ is 0.0605 and occurs in the cell in which the intruder is descending at a vertical rate between 77.34 and 78.13 ft/s and is located between 656.25 and 664.06 ft below the ownship. The ownship is also descending rapidly between 99.22 and 100.0 ft/s. Therefore, we can guarantee that the probability of NMAC with respect to the dynamics model outlined in this work when following the neural network policy is less than 0.0605. Tuning parameters such as the online splitting thresholds and minimum cell size allowed us to tighten our bound on the

probability. The full scale analysis was run on 9 cores of an Intel Xeon 2.20 GHz CPU and took approximately 10 days. With more computational resources, it is likely that we could tighten this bound further.

7 Related work

The complexity of deep neural networks makes their performance difficult to verify, and previous research has addressed this problem from multiple perspectives. One perspective involves decreasing the complexity of deep neural networks by extracting simpler policies that are easier to verify (Bastani et al. 2018; Koul et al. 2018; Carr et al. 2020). For instance, Bastani et al. (2018) outline a method to learn decision tree policies from neural networks trained used deep reinforcement learning. Other works focus on extracting finite state controllers from recurrent neural network policies (Koul et al. 2018; Carr et al. 2020). While these works simplify the verification problem, the extracted policies may be less effective and more difficult to store. In fact, Julian et al. (2019a) found that neural network policies performed better and required significantly less storage than decision trees when applied to the aircraft collision avoidance problem.

Another perspective on ensuring the safety of neural networks involves verifying their policies directly without the need to extract a simpler policy. Recent work has used formal methods to verify input-output properties of neural networks (Katz et al. 2017, 2019; Wang et al. 2018; Tjeng et al. 2017). Liu et al. (2021) provide an overview of neural verification methodologies that incorporate techniques from reachability, optimization, and search. In the context of neural network controllers, these tools can provide guarantees on the actions that the network could output in a given region of the state space. This methodology was used to prove properties of neural networks trained on an early prototype of ACAS Xu, the version of ACAS X developed for unmanned aircraft (Katz et al. 2017; Owen et al. 2019). An example property that should hold for the ACAS Xu networks is that the system should always issue an alert if an intruder is directly in front of the ownship. Katz et al. (2017) use the Reluplex algorithm for neural network verification to prove this property, along with a number of other intuitive properties that must hold for safe operation. Other verification tools have proven the same properties as a benchmark (Wang et al. 2018; Liu et al. 2021).

While neural network verification tools provide guarantees on the performance of neural network controllers at a single instance in time, other work has extended these tools to evaluate their performance over time using a model of the system dynamics. Julian et al. (2019b) define "safeable" regions for each action, in which even the worst case trajectory created by following the action could still be safe. This worst-case analysis, however, does not consider whether states are reachable when following the neural network policy, so states that it flags as unsafe may never be reached in the first place.

To better approximate the true closed-loop system, previous work has used various forms of reachability analysis (Julian and Kochenderfer 2019b, a; Huang et al. 2019; Xiang and Johnson 2018; Xiang et al. 2018, 2019; Dutta et al. 2019; Ivanov et al. 2019; Clavière et al. 2020; Lopez et al. 2021). These reachability approaches typically involve combining work in neural network verification with existing reachability methods from fields such as ordinary differential equations and hybrid systems. Other approaches involve combining the neural network controller with a neural network representation of the dynamics and performing the neural network verification on the entire closed-loop system (Sidrane and Kochenderfer 2019; Akintunde et al. 2018).

Previous works have applied reachability analysis to evaluate the closed-loop performance of aircraft collision avoidance neural networks (Lopez et al. 2021; Clavière et al. 2020; Julian and Kochenderfer 2019a; Akintunde et al. 2020). While Lopez et al. (2021) and Clavière et al. (2020) are able to show that the aircraft will not collide in a given set of scenarios, they do not take into account any uncertainty in the dynamics of either aircraft. Julian and Kochenderfer (2019a) and Akintunde et al. (2020) account for uncertainty in the dynamics by allowing the ownship and intruder to follow a range of accelerations. While these approaches are able to provide deterministic guarantees that the aircraft will not collide, they require strong assumptions on the dynamics of the aircraft for the conclusion to hold. As these assumptions are relaxed, there is likely to exist a set of adversarial intruder and ownship accelerations that could result in a collision. Even though the aircraft are unlikely to follow this acceleration pattern, there is still a nonzero probability that they will follow it, and the reachability analysis would conclude that the system is unsafe.

In this work, we adapt methods in probabilistic model checking to provide probabilistic guarantees for neural network controllers (Baier and Katoen 2008; Lahijanian et al. 2011; Bouton et al. 2020; D'argenio et al. 2001). There has been extensive previous work on model checking for finite state MDPs (Baier and Katoen 2008; Lahijanian et al. 2011). MDP model checking can be used to determine the probability of satisfying an LTL formula when following a given policy from any state in the state space. However, these methods are limited to problems with discrete states and cannot be applied directly to continuous neural network policies. This work therefore extends these methods to handle continuous states. To provide guarantees, we ensure that the probability estimates using our method represent an overapproximation of the true probability of failure.

8 Conclusion

In this work, we have introduced an approach to generate probabilistic safety guarantees on a neural network controller and applied it to an open source collision avoidance system inspired by the ACAS X neural networks. We demonstrated how techniques in MDP model checking could be applied to verify an overapproximated neural network policy obtained using a neural network verification tool. We identified the major sources of overapproximation error in the model checking process and developed both offline and online error reduction techniques to address them.

Our adaptive verification technique efficiently partitions the input space to obtain an overapproximated neural network policy that minimizes overapproximation error. This method can be used outside of the context of model checking to analyze neural network policies and detect decision boundaries. By combining the adaptive verification results with online splitting heuristics inspired by MDP state abstraction, we are able to provide meaningful probabilistic safety guarantees that follow trends shown in both Monte Carlo analysis and model checking analysis performed on the lookup table that the neural network approximates.

The probabilistic dynamics model used in this work represents a conservative aircraft response model. In the future, other aircraft response models will be analyzed to determine the effect of the model selection on the probabilities. Furthermore, the results can be used to determine unsafe areas of the state space where the neural network policy may need adjustments. For example, at the outset of this work, the model checking method was able to easily detect a bounds error in the neural network policy generation that may have been

otherwise easy to overlook. Although we used the method to determine the probability of an NMAC in this work, the general formulation allows us to determine the probability of satisfying any LTL specification. Future work will explore both generating guarantees on other aspects of the aircraft collision avoidance problem as well as for other safety-critical applications such as autonomous driving. The methodology presented here represents a step toward the ability to verify the performance of neural network controllers for use in safety-critical environments.

Funding This research was supported by National Science Foundation Graduate Research Fellowship under Grant No. DGE-1656518. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Data availability The neural networks used in this research can be found in the "networks" folder of the repository located at https://github.com/sisl/AdaptiveVerification.

Code availability The code for the adaptive verification portion of the work can be found at https://github.com/sisl/AdaptiveVerification, and the code for the model checking is located at https://github.com/sisl/NeuralModelChecking. The repository used to generate the networks used in this work is at https://github.com/sisl/VerticalCAS.

Declarations

Conflict of interest The authors have no conflicts of interest to declare that are relevant to the content of this article.

References

Akintunde, M., Lomuscio, A., Maganti, L., & Pirovano, E. (2018). Reachability analysis for neural agent-environment systems. In *International conference on principles of knowledge representation and reasoning*, pp 184–193.

Akintunde, M.E., Botoeva, E., Kouvaros, P., & Lomuscio, A. (2020). Formal verification of neural agents in non-deterministic environments. In *AAMAS*, pp 25–33.

Baier, C., & Katoen, J. P. (2008). *Principles of model checking*. MIT Press.

Bastani, O., Pu, Y., & Solar-Lezama, A. (2018). Verifiable reinforcement learning via policy extraction. arXiv preprint arXiv:180508328.

Bellman, R. (1952). On the theory of dynamic programming. *Proceedings of the National Academy of Sciences of the United States of America, 38*(8), 716.

Bouton, M. (2020). Safe and scalable planning under uncertainty for autonomous driving. Ph.D. thesis, Stanford University, https://purl.stanford.edu/dy440kv7606.

Bouton, M., Tumova, J., & Kochenderfer, M.J. (2020). Point-based methods for model checking in partially observable Markov decision processes. In *AAAI conference on artificial intelligence (AAAI)*, https://aaai.org/Papers/AAAI/2020GB/AAAI-BoutonM.9314.pdf.

Carr, S., Jansen, N., & Topcu, U. (2020). Verifiable rnn-based policies for pomdps under temporal logic constraints. arXiv preprint arXiv:200205615.

Clavière, A., Asselin, E., Garion, C., & Pagetti, C. (2020). Safety verification of neural network controlled systems. arXiv preprint arXiv:201105174.

Dutta, S., Chen, X., & Sankaranarayanan, S. (2019). Reachability analysis for neural feedback systems using regressive polynomial rule inference. In *ACM international conference on hybrid systems: Computation and control*, pp 157–168.

D'argenio, P.R., Jeannet, B., Jensen, H.E., & Larsen, K.G. (2001). Reachability analysis of probabilistic systems by successive refinements. In *Joint international workshop on process algebra and probabilistic methods, performance modeling and verification*, Springer, pp 39–56.

Huang, C., Fan, J., Li, W., Chen, X., & Zhu, Q. (2019). ReachNN: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS), 18*(5s), 1–22.

Ivanov, R., Weimer, J., Alur, R., Pappas, G.J., & Lee, I. (2019). Verisig: Verifying safety properties of hybrid systems with neural network controllers. In *ACM international conference on hybrid systems: Computation and Control*, pp 169–178.

Julian, K.D., & Kochenderfer, M.J. (2019a). Guaranteeing safety for neural network-based aircraft collision avoidance systems. In *Digital avionics systems conference (dasc)*. https://doi.org/10.1109/DASC43569.2019.9081748. arXiv:org/abs/1912.07084.

Julian, K.D., & Kochenderfer, M.J. (2019b). A reachability method for verifying dynamical systems with deep neural network controllers (1903.00520), arXiv:org/abs/1903.00520.

Julian, K. D., Lopez, J., Brush, J. S., Owen, M. P., & Kochenderfer, M. J. (2016). Policy compression for aircraft collision avoidance systems. *Digital Avionics Systems Conference (DASC)*. https://doi.org/10.1109/DASC.2016.7778091.

Julian, K. D., Kochenderfer, M. J., & Owen, M. P. (2019a). Deep neural network compression for aircraft collision avoidance systems. *AIAA Journal of Guidance, Control, and Dynamics, 42*(3), 598–608. https://doi.org/10.2514/1.G003724.

Julian, K.D, Sharma, S., Jeannin, J.B., & Kochenderfer, M.J. (2019b). Verifying aircraft collision avoidance neural networks through linear approximations of safe regions. In *AIAA spring symposium*, arXiv:org/abs/1903.00762.

Katz, G., Barrett, C., Dill, D.L., Julian, K.D., & Kochenderfer, M.J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer-aided verification*. arXiv:org/abs/1702.01135.

Katz, G., Huang, D.A., Ibeling, D., Julian, K.D., Lazarus, C., Lim, R., Shah, P., Thakoor, S, Wu, H., & Zeljić, A., Dill, D. L. (2019). The marabou framework for verification and analysis of deep neural networks. In *International conference on computer aided verification*. Springer, pp 443–452.

Kochenderfer, M. J. (2015). *Decision making under uncertainty: Theory and application*. MIT Press.

Kochenderfer, M.J., & Chryssanthacopoulos, J. (2011). Robust airborne collision avoidance through dynamic programming. Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371.

Kochenderfer, M. J., Holland, J. E., & Chryssanthacopoulos, J. P. (2012). *Next-generation airborne collision avoidance system*. Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States: Tech. rep.

Koul, A., Greydanus, S., & Fern, A. (2018). Learning finite state representations of recurrent policy networks. arXiv preprint arXiv:181112530.

Lahijanian, M., Andersson, S., & Belta, C. (2011). Control of Markov decision processes from PCTL specifications. In *American control conference*, IEEE, pp 311–316.

Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., & Kochenderfer, M. J. (2021). Algorithms for verifying deep neural networks. *Foundations and Trends in Optimization, 4*(3–4), 244–404. https://doi.org/10.1561/2400000035.

Lopez, D.M., Johnson, T., Tran, H.D., Bak, S., Chen, X., & Hobbs, K.L. (2021). Verification of neural network compression of ACAS Xu lookup tables with star set reachability. In *AIAA Scitech Forum*. p 0995.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *Nature, 518*(7540), 529–533.

Munos, R., & Moore, A. (2002). Variable resolution discretization in optimal control. *Machine Learning, 49*(2–3), 291–323.

Olson, W.A. (2015). Airborne collision avoidance system X. Tech. rep., Massachusetts Institute of Technology-Lincoln Laboratory Lexington United States.

Owen, M.P., Panken, A., Moss, R., Alvarez, L., & Leeper, C. (2019). ACAS Xu: Integrated collision avoidance and detect and avoid capability for UAS. In *IEEE/AIAA digital avionics systems conference (DASC)*, pp 1–10.

Pan, Y., Cheng, C.A., Saigol, K., Lee, K., Yan, X., Theodorou, E., & Boots, B. (2017). Agile autonomous driving using end-to-end deep imitation learning. arXiv preprint arXiv:170907174.

Sidrane, C., & Kochenderfer, M.J. (2019). OVERT: Verification of nonlinear dynamical systems with neural network controllers via overapproximation. In *Workshop on safe machine learning, international conference on learning representations*.

Tjeng, V., Xiao, K., & Tedrake, R. (2017). Evaluating robustness of neural networks with mixed integer programming. arXiv preprint arXiv:171107356.

Wang, S., Pei, K., Whitehouse, J., Yang, J., & Jana, S. (2018). Formal security analysis of neural networks using symbolic intervals. In *USENIX security symposium*., pp 1599–1614, https://www.usenix.org/conference/usenixsecurity18/presentation/wang-shiqi.

Xiang, W., & Johnson, T.T. (2018). Reachability analysis and safety verification for neural network control systems. arXiv preprint arXiv:180509944.

Xiang, W., Tran, H.D., Rosenfeld, J.A., & Johnson, T.T. (2018). Reachable set estimation and safety verification for piecewise linear systems with neural network controllers. In *American control conference*, pp 1574–1579.

Xiang, W., Lopez, D.M., Musau, P., & Johnson, T.T. (2019). Reachable set estimation and verification for neural network models of nonlinear dynamic systems. In *Safe, autonomous and intelligent vehicles*. Springer, pp 123–144.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.