University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Computer Science

Deep Learning Meets Formal Methods: A Certified Neural Network Synthesis Approach for Airborne Collision Avoidance

Supervisor
Matteo Zavatteri
University of Padova

Co-supervisor
Davide Bresolin
University of Padova

Master Candidate
Enrico Muraro

Student ID
1238899

Academic Year
2024-2025

Dedico questa pagina della mia tesi a tutte le persone che mi hanno aiutato durante questo percorso. Vorrei ringraziare il mio relatore, il professore Zavatteri, per la pazienza e i consigli durante il lavoro della tesi. Grazie anche al mio correlatore, il professore Bresolin, per l'aiuto nella scelta dell'argomento e la disponibilità fin dall'inizio.
Un ringraziamento speciale ai miei genitori e mio fratello per l'infinito supporto, senza i quali non sarei mai arrivato qua.
Un grazie infine a tutti gli amici, online e non, e ai colleghi di corso conosciuti in questi anni.

Abstract

Deep neural networks (DNNs) are being deployed in countless real-world applications, including safety-critical domains such as autonomous vehicles and collision avoidance systems in unmanned aircraft. Due to the unpredictability of the network in edge cases, the use of formal verification techniques has become a necessity to certify safety specifications. However, when the network is not certified for a certain property, it is also essential to have a method for adjusting its behavior. Hence, we focus on the synthesis of certified DNNs, that is, how to automatically build a neural network that is guaranteed to satisfy the required properties. We use a *Counter-Example Guided Inductive Synthesis* (CEGIS) loop, alternating phases of training, formal verification, and training data generation from counterexamples, to synthesize certified networks without impacting their classification accuracy. Finally, we evaluate this approach on a DNN-based collision avoidance system, ACAS Xu, demonstrating the feasibility of this synthesis method in practical applications.

Contents

Listing of figures

Listing of tables

1

Background and related works

1.1 Introduction

With the growing use of Deep Neural Networks (DNNs) in safety-critical fields such as self-driving vehicles and collision prevention systems, it is becoming essential to formally verify the safety properties of these DNNs. In these applications errors or unexpected corner cases can easily lead to extensive damage or, in the worst case, to loss of human lives. Many current methods for testing the safety of DNNs focus on identifying adversarial examples [1], but fail to offer formal assurances regarding the non-existence of such adversarial examples. Moreover, DNNs have been shown to respond in unexpected and incorrect ways to minor disturbances in their inputs [2]. Hence, formal verification techniques must be able to exhaustively explore the input space to ensure that the behavior of the network aligns with the desired properties.

For this reason, many of the current approaches introduce Satisfiability Modulo Theories (SMT) or Mixed Integer Linear Programming (MILP) solvers to formally check safety properties. Tools such as Reluplex [3] and Marabou [4] use specialized versions of the SIMPLEX method to verify ReLU activated neural networks. ReluVal [5], avoids the complexity of SMT solvers by using symbolic interval analysis to estimate the bounds of the output and verify network properties.

However, when the property is not respected, we also need a way to synthesize a neural network that is guaranteed to respect it. The problem of synthesizing a program that satisfies a

certain property has been studied even outside of neural networks, ranging from synthesizing loop bodies from pre/conditions [6], loop-free programs [7], or Excel macros from examples [8]. It is formalized as the *syntax-guided synthesis problem* (SyGuS) [9], and it even sparked a competition [10] on a large collection of benchmarks with the objective of advancing research and tools on the subject. One of the most relevant methodologies involves exploiting the counterexample given by the verifier to iteratively improve the original network.

The *Counter Example Guided Inductive Synthesis* (CEGIS) workflow, initially proposed in [11], alternates verification and training of the network to effectively synthesize a DNN guaranteed to respect a certain logical property. This method has demonstrated great success in tools like SpecRepair [12], and work by [13] introduces some sufficient conditions to guarantee termination.

1.2 Neural Networks

For the purposes of this thesis we consider only feedforward, ReLU activated neural networks. A neural network can be represented as a weighted graph in which each node performs an operation. The nodes are grouped into different layers depending on the function:

- input layer: simply passes the value of the input x, an array of real numbers, to the next layer.
- hidden layer: present multiple times for Deep Neural Networks, each node performs some mathematical operation on its inputs followed by an activation function, ReLU in this case. Each node also has an associated *bias*. The output is then forwarded to all nodes in the next layer.
- output layer: the final layer of the network. Each node has a bias and performs a mathematical operation on its inputs, but usually does not have an activation function.

In particular, the graph for feedforward neural networks does not contain loops, and each layer feeds directly into the next one. All layers are *fully connected* since each node receives all the outputs of the previous layer.

In practice, the graph represents a function from vectors of real numbers to vectors of real numbers.

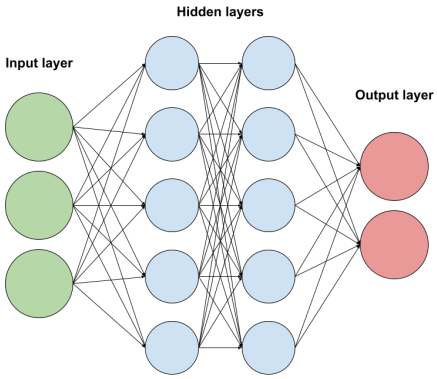


Figure 1.1: A fully connected feedforward neural network with 3 input nodes, 2 hidden layers with 5 nodes each, 2 output nodes

1.3 Linear Real Arithmetic

The theory of *linear real arithmetic* (LRA) is particularly fitting in the context of encoding a neural network and its properties, as it is capable of representing many neural network operations [14].

LRA is built with atoms in the form of a linear inequality:

$$\sum_{i=1}^{n} c_i x_i + b \bowtie 0 \tag{1.1}$$

Where $c_i, b \in \mathbb{R}, \bowtie \in \{<, \leq\}$ and $\{x_i\}$ is a fixed set of variables. It should be noted that $>$ and $\geq$ can be represented using $<$ and $\leq$, respectively. An equality $x = 0$ can be represented as:

$$x \geq 0 \wedge x \leq 0 \tag{1.2}$$

Similarly, an inequality $x \neq 0$ can be written as:

$$x < 0 \vee x > 0 \tag{1.3}$$

A hidden node in the neural network can then be encoded as its bias plus the weighted sum of its inputs, in conjunction with the ReLU activation of its output.

Formally, let N be a feedforward neural network , with d layers, where 1 is the input layer, d is the output layer and $2, \ldots, d-1$ are the hidden layers. Let l_i be the size of the layer i, for

each node $v_{i,j}$, where $1 < i < d$ and $1 \leq j \leq l_i$, we add two auxiliary variables $v_{i,j}^{in}$ and $v_{i,j}^{out}$ to represent, respectively, the input and output values of the node.

The function that represents the computation of a hidden node $v_{i,j}$ can then be encoded with the following conjuncts:

$$v_{i,j}^{in} = b_{i,j} + \sum_{k=1}^{l_i - 1} w(v_{i-1,k}, v_{i,j}) \cdot v_{i-1,k}^{out} \tag{1.4}$$

$$v_{i,j}^{out} = max(0, v_{i,j}^{in}) \tag{1.5}$$

where $w(a, b)$ is the weight of the arc between the nodes a and b.

The equality $a = max(0, b)$ that encodes the ReLU activation function on the output value of the node can be written in LRA as:

$$a \geq 0 \wedge a \geq b \wedge (a = 0 \vee a = b) \tag{1.6}$$

Similarly, we can encode the output nodes using the same formula, but without applying the ReLU activation to the output value of the node. Hence, for each output node $v_{d,j}$, we add the following conjuncts:

$$v_{d,j}^{in} = b_{d,j} + \sum_{k=1}^{l_d - 1} w(v_{d-1,k}, v_{d,j}) \cdot v_{d-1,k}^{out} \tag{1.7}$$

$$v_{d,j}^{out} = v_{d,j}^{in} \tag{1.8}$$

The input layer does not perform any calculations, so the formula can simply forward the input to the output of the node. For each input node $v_{1,j}$, we add the following conjuncts:

$$v_{1,j}^{out} = v_{1,j}^{in} \tag{1.9}$$

Let $x = (x_1, ..., x_{l_1})$ be the vector that represents the input nodes and $y = (y_1, ..., y_{l_d}$ the vector that represents the output nodes. The final formula that encodes the entire neural network N is the conjunction of all the equations obtained using the method explained above, with the auxiliary variables $v_{i,j}^{in}$ and $v_{i,j}^{out}$ existentially quantified.

The resulting LRA formula $F_N(x, y)$ is true if and only if $y = N(x)$, which means that the encoding is both sound and strict, as proven by [14], and correctly captures the behavior of N.

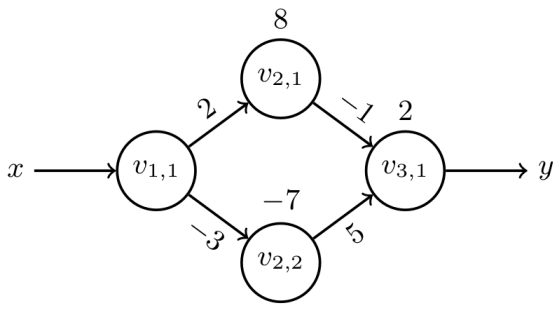


Figure 1.2: Small neural network with one hidden layer

1.3.1 Encoding example

We demonstrate the LRA encoding of the neural network depicted in Figure 1.2 as a small example. The neural network has one input node ($v_{1,1}$), one output node ($v_{3,1}$ and two nodes in the hidden layer ($v_{2,1}$ and $v_{2,2}$) with ReLU activation functions. The weights and biases are shown over the arcs and nodes respectively.

Each node, except for the input node, computes the weighted sum of its inputs plus its own bias. Including the ReLU activation function for the hidden layers, the functions computed by $v_{2,1}$ and $v_{2,2}$ are:

$$\begin{aligned} v_{2,1} &:= max(0, 2x + 8) \\ v_{2,2} &:= max(0, -3x - 7) \end{aligned} \tag{1.10}$$

The function computed by this network is then the following:

$$N(x) := 2 - (max(0, 8 + 2x)) + 5(max(0, -7 - 3x)) \tag{1.11}$$

We can use the LRA encoding described in the previous section to encode this network. First, we encode the input simply by forwarding it to the output of its auxiliary variable:

$$x = v_{1,1}^{out}$$

Then, the hidden nodes are encoded as such:

$$v_{2,1}^{in} = 8 + 2v_{1,1}^{out} \ \wedge \ v_{2,1}^{out} = max(0, v_{2,1}^{in}) \text{ for the hidden node } v_{2,1}$$
$$v_{2,2}^{in} = -7 - 3v_{1,1}^{out} \ \wedge \ v_{2,2}^{out} = max(0, v_{2,2}^{in}) \text{ for the hidden node } v_{2,2}$$

The output node is encoded similarly, but without ReLU activation. Instead, its output is forwarded to the output variable y:

$$v_{3,1}^{in} = 2 - v_{2,1}^{out} + 5v_{2,2}^{out} \ \wedge \ y = v_{3,1}^{in}$$

Finally, by existentially quantifying the auxiliary variables, we obtain the following LRA formula depending only on the input x and output y:

$$\begin{aligned} F_N(x,y) := \exists v_{1,1}^{out}, v_{2,1}^{in}, v_{2,1}^{out}, v_{2,2}^{in}, v_{2,2}^{out}, v_{3,1}^{in} (x = v_{1,1}^{out} \wedge \\ v_{2,1}^{in} = +8 + 2v_{1,1}^{out} \wedge v_{2,1}^{out} = max(0, v_{2,1}^{in}) \wedge \\ v_{2,2}^{in} = -7 - 3v_{1,1}^{out} \wedge v_{2,2}^{out} = max(0, v_{2,2}^{in}) \wedge \\ v_{3,1}^{in} = 2 - v_{2,1}^{out} + 5v_{2,2}^{out} \wedge y = v_{3,1}^{in}) \end{aligned} \tag{1.12}$$

This encoding perfectly captures the mathematical function of the network in a logical formula.

1.4 Formal Verification

The properties of the network that we seek to verify must also be expressible in LRA. They can be defined as a precondition on the input of the network and a postcondition that specifies what we expect to hold true.

More precisely, let $F_{pre}(x)$ be the precondition for the input variables and $F_{post}(x, y)$ the postcondition for the input and output variables, both defined as LRA formulas, then the properties will be of the form

$$F_{pre}(x) \Rightarrow F_{post}(x, y) \tag{1.13}$$

The network is certified when for all possible inputs that satisfy the precondition $F_{pre}(x)$ the postcondition $F_{post}(x, y)$ is also satisfied. This means that we certify a network by checking if the following formula is valid:

$$F_{N,P}(x, y) := F_{pre} \wedge F_N(x, y) \Rightarrow F_{post} \tag{1.14}$$

where $F_N(x, y)$ is the LRA encoding of the network.

We say that a property P is realizable if there exists a function $G : X \mapsto Y$ such that $F_{pre}(x) \Rightarrow F_{post}(x, G(x))$ is true for all $x \in X$. From now on, we consider only realizable properties.

Since an LRA formula is valid if and only if its negation is unsatisfiable, an equivalent way to certify a network is to check the satisfiability of $\neg F_{N,P}(x, y)$. This results in two possible cases:

- $\neg F_{N,P}(x, y)$ is satisfiable, which means that there is at least one counterexample where the input of N satisfies the precondition of the property P, but the postcondition is false.
- $\neg F_{N,P}(x, y)$ is unsatisfiable, which means that the property P holds for all possible inputs of N that satisfy the precondition and the network is certified.

$\neg F_{N,P}(x, y)$ can be written more explicitly as:

$$\neg F_{N,P}(x, y) := \neg(F_{pre} \wedge F_N(x, y) \Rightarrow F_{post}) \equiv F_{pre} \wedge F_N(x, y) \wedge \neg F_{post} \tag{1.15}$$

The advantage of this approach is that we can use any state-of-the-art SMT solver for LRA to check the satisfiability of $\neg F_{N,P}(x, y)$, and consequently to certify the properties of a neural network.

1.4.1 Property verification example

Consider the following property P for the network N described above in Figure 1.2.

P: *If the input is less than or equal to zero, then the output is also less than or equal to zero*

We want to verify whether this property is always true for all possible negative or zero inputs. First, the property is rewritten in LRA form:

$$x \leq 0 \Rightarrow y \leq 0 \tag{1.16}$$

Here $x \leq 0$ is the precondition of the property ($F_{pre}(x)$), while $y \leq 0$ is the postcondition ($F_{post}(x, y)$).

We combine the property with the LRA encoding of N obtained before (1.12) and obtain the following formula:

$$F_{N,P}(x, y) := (F_N(x, y) \wedge x \leq 0) \Rightarrow y \leq 0 \tag{1.17}$$

In this case, N is not certified for P, since the negation of the formula

$$\neg F_{N,P}(x, y) := F_N(x, y) \wedge x \leq 0 \wedge y > 0 \tag{1.18}$$

is satisfiable. That is, we can find an assignment of real values α for all variables in x and y such that formula $\neg F_{N,P}(x, y)$ is true.

For example, one possible solution is $\alpha(x) = -3, \alpha(y) = 10$ since $N(-3) = 10$. This α is a valid counterexample showing that the property P is not satisfied by N.

2

Certifying Neural Networks

2.1 Certification Process

The certification process follows the *Counter Example Guided Inductive Synthesis (CEGIS)* workflow shown in figure 2.1. Starting from a dataset of input-output pairs divided into training, validation and test sets, we train a neural network N. We then encode N into a suitable *Constraint Satisfaction Problem (CSP)*. In order to formally verify that N satisfies a certain property P we can look for a counterexample, that is, a specific input on the neural network whose output does not satisfy P. By adding $\neg P$ to the *CSP* and solving the satisfaction problem in the formal verification step, we can obtain two possible results:

1. The CPS is unsatisfiable, which means that there is no possible counterexample, and N is certified for P
2. The CPS is satisfiable, which means that the CPS solution is the counterexample we were looking for, and N is not certified for P

In the case where we find a counterexample with input x whose output $N(x)$ does not satisfy P, we can use this information to generate one or more input-output pairs to add to the training dataset and train the network again. These points are obtained using the input x from the counterexample and 'repairing' it by substituting the output with that expected to satisfy property P

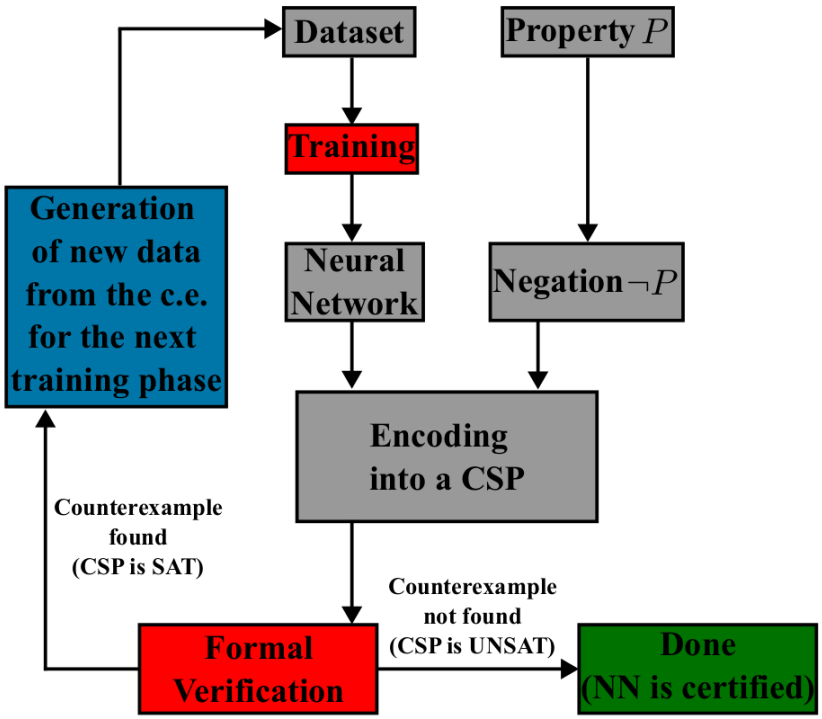


Figure 2.1: High level view of the Counter Example Guided Inductive Synthesis (CEGIS) workflow

This workflow can be repeated until no counterexample is found. In that case, the process is complete and N is considered certified, that is, for any possible input, N will always respect the property P.

More formally, the problem statement becomes:

- given a dataset D of input-output pairs
- given a realizable property P

synthesize a ReLU neural network N such that:

- N is certified for P, and
- the accuracy of N is preserved as much as possible with respect to the accuracy of network trained on D only.

2.2 Counterexample repair and data generation

If a network N is not certified for a property P, we could try to correct the behavior of N by retraining using the original training dataset D and check if, after training, N is now certified for P. However, this process could take several iterations to complete or, in the worst case, it may not terminate when D does not contain any training point that corresponds to the property P.

The high-level idea is then to add specific training points to the original dataset D before retraining the network. Ideally, at each iteration, these points will move the network closer to satisfying P and the process will end with a network formally certified for P.

During formal verification, when $\neg F_{N,P}(x, y)$ is satisfiable, we also obtain a satisfying assignment α. In this case N is not certified for P, and α corresponds to the counterexample where $F_{pre}(x)$ is true, but $F_{post}(x, y)$ is false.

To obtain a new training point for the network, we can repair the counterexample α by keeping the same input values and changing the output value in a way that satisfies P.

In other words, we want to build a new assignment α' such that:

1. The assignment of values to the input variables does not change: for each $i = 1, \ldots, n$, it holds that $\alpha'(x_i) = \alpha(x_i)$.
2. α' makes $F_{post}(x, y)$ true.

Starting from the output values of α, we can add a new slack variable s_i to each output value $\alpha(y_i)$. These variables represent how each output value needs to change, by adding or removing a certain amount, in order to satisfy $F_{post}(x, y)$.

For example, suppose that we have $F_{post}(x, y) := y_1 \geq 0$ as a postcondition and find a counterexample α where $(\alpha(x_1) = 1, \alpha(y_1) = -5)$. In order to repair the counterexample, we need to find a value for s_1 such that $-5 + s_1 \geq 0$. In this case, any value for s_1 that is greater than or equal to 5 will satisfy the original postcondition.

However, in order to maintain the accuracy of the network in the test set, we want to keep the values of the repaired counterexample α' as close as possible to the original data distribution,. In the previous example, we would rather use $s_1 = 5$ than $s_1 = 1,000,000$, even if both would satisfy $-5 + s_1 \geq 0$, since large values may negatively affect the accuracy of the network on similar inputs after training. In other words, we want to minimize the difference between α and α'.

In this case, the generated training point that will be added to D then becomes $(\alpha(x_1), \alpha(y_1) + 5) = (1, -5 + 5) = (1, 0)$.

Formally, we can describe the process as follows: Let N be a neural network and P a property on N, suppose that $\neg F_{N,P}(x, y)$ is satisfiable and α is a satisfying assignment. Let $s = (s_1, ..., s_m)$ be a vector of new variables that do not appear in $F_{post}(x, y)$.

We can define a new LRA formula that utilizes the values in α and the new slack variables s as such:

$$F'_{post}(s) := F_{post}(x, y)[\alpha(x_1)/x_1, ..., \alpha(x_n)/x_n, \alpha(y_1) + s_1/y_1, ..., \alpha(y_m) + s_m/y_m] \tag{2.1}$$

where $F[t_1/x_1, ..., t_k/x_k]$ is the simultaneous multiple substitution that replaces every occurrence of x_i with the corresponding t_i.

Any assignment β that satisfies $F'_{post}(s)$ can be used to create a repaired counterexample α' as follows:

1. For each $i = 1, ..., n$, $\alpha'(x_i) = \alpha(x_i)$
2. For each $i = 1, ..., m$, $\alpha'(y_i) = \alpha(y_i) + \beta(s_i)$

By construction, any α' created using this method will satisfy the original postcondition $F_{post}(x, y)$ and can be used to create the new training point (x', y') in the data set D, where:

- $x' = (\alpha'(x_1), ..., \alpha'(x_n))$
- $y' = (\alpha'(y_1), ..., \alpha'(y_m))$

The α' that minimizes the distance from the counterexample α' can be computed by solving the following quadratic programming problem:

$$min \sum_{i=1}^{m} s_i^2 \quad \text{subject to} \quad F'_{post}(s) \tag{2.2}$$

By adding a single data point to the training set at the time, the certification process might require many iterations. However, we can generate additional k points by sampling around α' by varying the values by a small amount ε, while still making sure that it satisfies $F_{post}(x, y)$. The values of the parameters k and ε need to be fine-tuned to each specific application.

2.3 Certifying multiple properties

In certain cases, we might need to certify multiple properties at the same time for a single neural network, each with different pre- and post-conditions.

Let j be the number of properties, each property can be indexed as such:

$$P^i := F^i_{pre}(x) \Rightarrow F^i_{post}(x,y) \quad \text{for each } i = 1, \ldots, j \tag{2.3}$$

When not all preconditions are independent, that is, multiple properties involve the same area in the input space, then it is possible to have cases where the network is impossible to certify. More formally, two properties are independent when the conjunction of their preconditions are unsatisfiable.

$$F^1_{pre}(x) \wedge F^2_{pre}(x) \quad unsat \tag{2.4}$$

For example, two properties with the same preconditions $F^1_{pre}(x) \equiv F^2_{pre}(x)$, but exact opposite postconditions $F^1_{post}(x,y) \equiv \neg F^2_{post}(x,y)$, will never be true at the same time, regardless of the shape or training of the network.

For the purposes of this thesis, we will consider only sets of properties that are independent.

To verify multiple properties at once, we could create a single LRA formula in this way:

$$F^{all}_N(x,y) = F_N(x,y) \wedge \bigvee_{i=1}^{j} (F^i_{pre}(x) \wedge \neg F^i_{post}(x,y)) \tag{2.5}$$

Then, by checking the satisfiability of $F^{all}_N(x,y)$, we obtain two possible cases:

- $F^{all}_N(x,y)$ is satisfiable; this means that one of the formulas $\neg F^i_{N,P^i}(x,y)$ is satisfied and we obtain a counterexample α^i.
- $F^{all}_N(x,y)$ is not satisfiable; this means that none of the single formulas $\neg F^i_{N,P^i}(x,y)$ is satisfiable, and the network is certified for all the properties.

The disadvantage of this approach is that while j properties are checked for satisfiability, we obtain only one counterexample α^i. This means that only one new training point can be generated at each iteration.

A better approach is to check the satisfiability of the individual formulas $F^i_{N,P^i}(x,y)$ for each property P^i. Consequently, we obtain up to j new counterexamples, one for each property

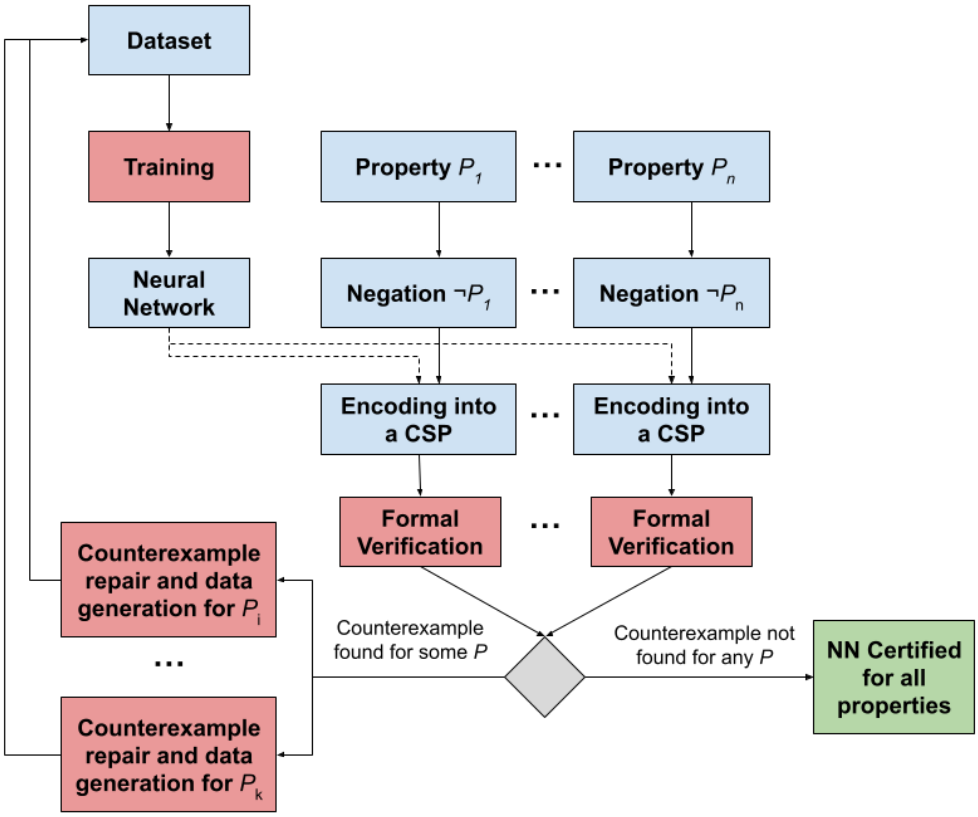


Figure 2.2: High level view of the CEGIS workflow for multiple properties

$\neg F^i_{N,P^i}(x,y)$ that is satisfied. The counterexamples can then be repaired to generate up to j new training points.

Finally, all new training points can be added to the dataset D before starting a new iteration of the CEGIS loop, reducing the number of training steps required. Each training point can also be sampled, as explained above, in order to increase the number of points added for each iteration.

Since the satisfiability of each property can be independently checked, this approach can also be easily computed in parallel, further improving the overall speed of the certification process.

The network is certified when every property $\neg F^i_{N,P^i}(x,y)$ is unsatisfiable. Therefore, it is unnecessary to fix counterexamples or retrain the networks.

3

Case Study: ACAS Xu

ACAS X is a family of Airborne Collision Avoidance Systems that uses large numeric lookup tables to help avoid airborne collisions, independently of ground-based equipment or air traffic control.

The version for large manned aircraft, ACAS Xa, offers pilots advisories and is already one of the international standards for aviation [15]. The version for unmanned aircraft, ACAS Xu, issues turn-rate advisories to remote pilots in order to avoid *near midair collisions* (NMACs), and practical demonstrations have been flown by NASA in 2014 [16].

Managing the size of the lookup tables can be challenging, especially for small unmanned aircraft. The lookup table contains the score of advisories for millions of different possible states, requiring hundreds of gigabytes of storage. To reduce the required size, it is possible to downsample the table, removing states in areas with minimal changes in scores, while keeping high accuracy in respect with the original table.

However, even after downsampling, the table requires over 2 gigabytes of floating point storage. Although this size can be easily managed with modern commercial hardware, aircraft-certified computer hardware is expensive, and a solution capable of running on legacy hardware is preferred.

In order to further reduce the storage requirements, the table can be approximated using a deep neural network to obtain the advisories instead. In this case, only the weights of the trained network would need to be stored, at the cost of a small amount of accuracy.

Simulations performed by [17] show that the run-time of the network closely matches that

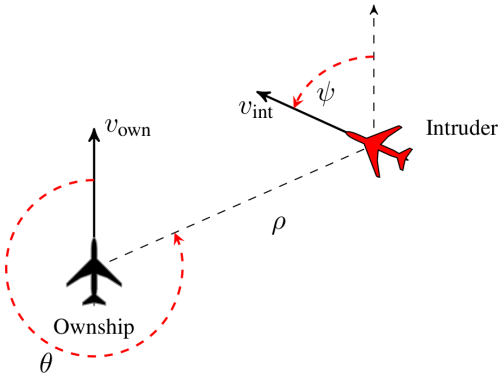


Figure 3.1: geometry of HorizontalCAS encounters

of the original table, ensuring equally fast responses to the external environment.

The primary concern of using a neural network is the uncertainty of where its behavior differs from the original table. The difference in accuracy could be in areas where the suggested advisory is irrelevant to the overall safety of the aircraft, for example, advising a turn slightly earlier than needed. However, in the worst case, the slight difference from the original table could result in an NMAC.

Neural networks are difficult to predict, and in safety-critical applications it is fundamental to guarantee that the network will perform correctly in all possible situations.

3.1 HorizontalCAS

Most ACAS X collision avoidance systems are still in development and are not publicly available. Hence, this thesis focuses on HorizontalCAS, an open source collision avoidance system inspired by early versions of ACAS Xa and ACAS Xu introduced by [18].

HorizontalCAS issues horizontal turn rate advisories to an aircraft, called the *ownship*, to avoid near midair collisions (NMACs) with another aircraft, called the *intruder*. NMACs occur when the intruder is within 500 ft horizontally and 100 ft vertically. The state space is composed of seven variables, summarized in table 3.1. For the purposes of this thesis, the ownship and intruder speeds are constant; ρ represents the distance between the two aircraft measured in feet; θ is the bearing angle to the intruder in degrees; ψ is the heading angle of the intruder in degrees; τ represents the time until the vertical separation between the ownship and the intruder is lost; s_{adv} is the previous advisory.

Hence, the horizontal geometry is defined using polar coordinates relative to the ownship's

position and heading. Assuming the ownship has $(0, 0)$ as Cartesian coordinates, the coordinates of the intruder would be $(\rho\, cos(\theta), \rho\, sin(\theta))$ respectively.

The collision avoidance system gives five possible advisories with different turning rates and directions. Clear of Conflict (COC) allows the ownship to turn freely, while weak or strong left (WL, SL), and weak or strong right (WR, SR) determine the recommended turn rate to avoid an NMAC. The advisories are summarized in Table 3.2.

Instead of a continuous variable, τ is discretized in eight possible states:

$$\tau \in \{0, 5, 10, 15, 20, 30, 40, 60\}$$

In this way, we can divide the problem into 40 different deep neural networks, one for each combination of τ and s_{adv}. Since the speeds are fixed to a constant value, the only remaining inputs for each DNN are (ρ, θ, ψ). The outputs for each DNN are the five possible advisories (COC, WL, WR, SL, SR), where the advisory with the highest output value is the one selected by the network. For the system to choose the correct network, the real value of τ is rounded down to the closest discretized value.

Variable	Description	Values
ρ (ft)	Range to intruder	[0, 56000]
θ (deg)	Bearing angle to intruder	[-180, 180]
ψ (deg)	Relative heading angle of int.	[-180, 180]
v_{own} (ft/s)	Ownship speed	200
v_{int} (ft/s)	Intruder speed	185
τ (s)	Time to loss of vert. separation	[0, 80]
s_{adv}	Previous advisory	See Table 3.2

Table 3.1: HorizontalCAS variables descriptions and values

Advisory	Description	Ownship Turn Rate
COC	Clear of Conflict	$[-1.5°/s, 1.5°/s]$
WL	Weak Left	$1.5°/s$
WR	Weak Right	$-1.5°/s$
SL	Strong Left	$3.0°/s$,
SR	Strong Right	$-3.0°/s$

Table 3.2: HorizontalCAS advisories description and values

3.2 Safety properties

In order to guarantee the safety of the DNN-approximated system, we need a way to define the safety properties and verify them. We selected eight safety properties, chosen as the ones most relevant from [5] and the angles are converted in radians:

- **Property φ_1:**
  - Description: If the intruder is near and approaching from the left, the network advises *strong right*.
  - Input constraints: $250 \leq \rho \leq 400, 0.2 \leq \theta \leq 0.4, 3.141592 \leq \psi \leq 3.141592 + 0.005$
  - Desired output property: the score for *strong right* is the maximal score.

- **Property φ_2:**
  - Description: If the intruder is directly ahead and is moving towards the ownship, the score for *COC* will not be maximal.
  - Input constraints: $1500 \leq \rho \leq 1800, 0.06 \leq \theta \leq 0.06, 3.10 \leq \psi \leq 3.14$
  - Desired output property: the score for *COC* is not the maximal score.

- **Property φ_3:**
  - Description: For a far away intruder (on the left), the network advises *COC*.
  - Input constraints: $36000 \leq \rho \leq 60760, 0.7 \leq \theta \leq 3.141592, 3.141592 \leq \psi \leq 3.141592 + 0.01$
  - Desired output property: the score for *COC* is maximal.

- **Property φ_4:**
  - Description: Even if the previous advisory was *weak right*, the presence of a nearby intruder will cause the network to output a *strong left* advisory instead.
  - Input constraints: $2000 \leq \rho \leq 5000, -3.141592 \leq \theta \leq -0.7, 3.141592 \leq \psi \leq 3.141592 + 0.01$
  - Desired output property: the score for *strong left* is maximal.

- **Property φ_5:**
  - Description: For a far away intruder, the network advises *COC* even if it is in front of the ownship.
  - Input constraints: $44000 \leq \rho \leq 50000, -1 \leq \theta \leq 1, 3.141592 \leq \psi \leq 3.141592 + 0.01$
  - Desired output property: the score for *COC* is maximal.

- **Property φ_6:**
  - Description: If the intruder is near and approaching from the right, the network advises *strong left*
  - Input constraints: $250 \leq \rho \leq 400, -0.4 \leq \theta \leq -0.2, 3.141592 \leq \psi \leq 3.141592 + 0.005$
  - Desired output property: the score for *strong left* is the maximal score.

- **Property φ_7:**
  - Description: For a far away intruder (on the right), the network advises *COC*.
  - Input constraints: $36000 \leq \rho \leq 60760, -3.141592 \leq \theta \leq -0.7, 3.141592 \leq \psi \leq 3.141592 + 0.01$
  - Desired output property: the score for *COC* is maximal.

- **Property φ_8:**
  - Description: Even if the previous advisory was *weak left*, the presence of a nearby intruder will cause the network to output a *strong right* advisory instead.
  - Input constraints: $2000 \leq \rho \leq 5000, 0.7 \leq \theta \leq 3.141592, 3.141592 \leq \psi \leq 3.141592 + 0.01$
  - Desired output property: the score for *strong left* is maximal.

3.3 Implementation

The code and complete implementation can be found in the following repository: https://github.com/EnricoMuraro/Certified-HorizontalCAS

3.3.1 Neural network implementation

We implemented the networks as fully-connected, feedforward DNNs with ReLU activations after each hidden layer. All forty networks share the same structure:

- 3 input nodes, corresponding to the continuous variables ρ, θ, and ψ in HorizontalCAS
- 5 hidden layers, decreasing in size. Each layer contains 128, 64, 32, 16, and 8 nodes, respectively.
- 5 output nodes, one for each advisory (COC, WL, WR, SL, SR)

The resulting network is small in number of parameters, which is important in order to obtain a meaningful compression compared to the original tables. However, the network is still able to accurately represent the original behavior. The memory required to store all forty networks is around 2MB, resulting in a compression factor of 1000 from the 2GB tables.

The training dataset has been generated by splitting the original decision table for HorizontalCAS into 40 different datasets, one for each network, based on the values of τ and s_{adv}. The implementation details of the data generation processes can be found in https://github.com/sisl/HorizontalCAS. Each training set contains 53,792 input-output pairs. The density is higher near the home and decreases with distance, as illustrated in Figure 3.2, as ensuring high accuracy is more crucial when the intruder is nearby than when it is distant.

The networks have been trained in Python, with the PyTorch library, using a machine with an NVIDIA 1050 ti GPU. Before training, all input and output values of the training set are normalized to have mean equal to zero and range equal to one, leading to a faster training process [19]. We use backpropagation with an Adam optimizer to train the weights of the network.

An important factor to consider is the loss function used during training. Although mean square error (MSE) is commonly used for its simplicity and ease of computation, we need to consider that the value between the correct advisory and the others is often small, and MSE treats any error in the advisories in the same way. Instead, we want to penalize the case where the correct advisory no longer has the maximum value, by increasing the loss function by a certain factor. Hence, we use an asymmetric version of MSE, shown in Figure 3.3, where the loss

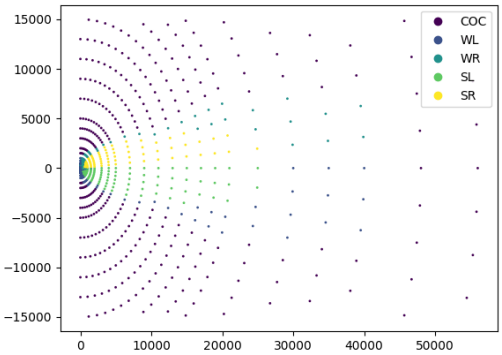


Figure 3.2: Scatter plot of training points for the network with $s_{adv} = COC, \tau = 0$

when the optimal advisory is underestimated is MSE increased by a factor of 4, while when it is overestimated the loss is simply MSE. Similarly, when a suboptimal advisory is underestimated the loss is MSE, meanwhile, when it is overestimated, the loss is increased.

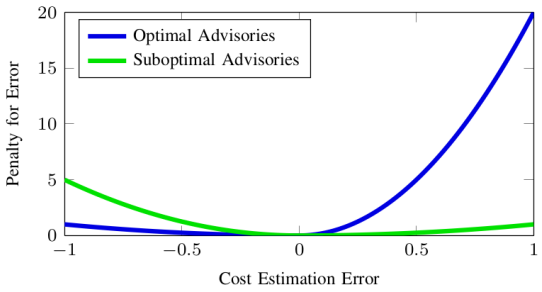


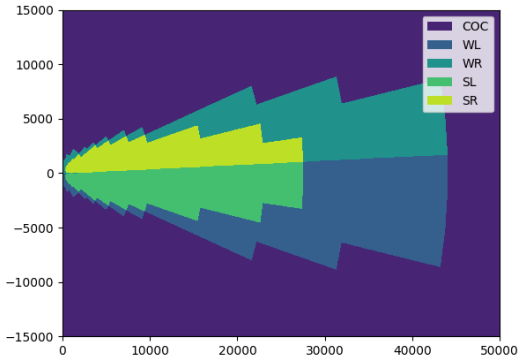
Figure 3.3: Asymmetric loss function used for training the network

The training set is shuffled and divided into batches of 128 samples for each training pass. Each network was trained for 1000 training epochs, for an average of 15 minutes to complete the training of a single network.

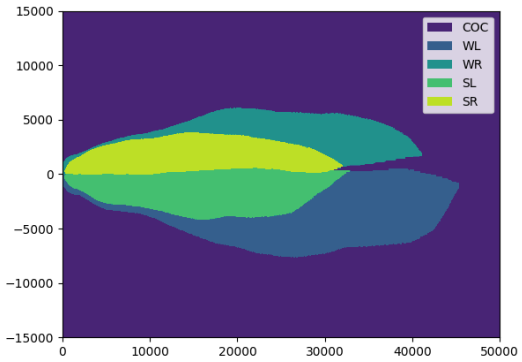
Finally, the trained networks are exported as ONNX[20] files, an open format used to represent machine learning models, to be compatible with other libraries that will be used during the certification process.

3.3.2 Certification implementation

The certification process is implemented following the CEGIS workflow described in Chap. 2. For the formal verification step, we use the state-of-the-art tool Marabou [21], instead of



(a) training advisories for network $\tau = 0, s_{adv} = COC$



(b) learned advisories for network $\tau = 0, s_{adv} = 00$ after training

Figure 3.4: training set behavior and learned behavior for network $\tau = 0, s_{adv} = COC$ for a head-on encounter scenario

implementing a custom verification algorithm. This is facilitated by the modular nature of the CEGIS workflow, which allows each step to function independently of implementation, enabling the integration of existing tools and solvers. When certifying a property, Marabou will return a counterexample if the property is not satisfied. Then we repair the counterexample by solving the quadratic programming problem described in Equation 2.2. For example, assuming that we are repairing a counterexample for property φ_1, the corresponding quadratic

programming problem would be:

$$
\begin{aligned}
&min \sum_{i=1}^{5} s_i^2 \quad subject\ to: \\
&y_1 + s_1 \leq y_5 + s_5 \\
&y_2 + s_2 \leq y_5 + s_5 \\
&y_3 + s_3 \leq y_5 + s_5 \\
&y_4 + s_4 \leq y_5 + s_5
\end{aligned}
\tag{3.1}
$$

where y_i corresponds to each variable of the network output in the following order:

$$y_1 = COC,\ y_2 = WL,\ y_3 = WR,\ y_4 = SL,\ y_5 = SR$$

Here, the property requires y_5 as the maximal score in order for it to be a correct training point, while the input remains identical to that provided in the counterexample. The solution can be found using any existing quadratic programming solver; for our implementation, we chose Gurobi [22].

Subsequently, the repaired counterexample is sampled in order to generate 10 training points at a time. We sample by changing the output values by a random small amount between 0 and 0.05, while checking that the result still satisfies the property.

Finally, the 10 newly sampled points are added to the original training set and the network is re-trained. Once no counterexamples are found, the network is formally certified for that specific property.

Unlike the initial training process, in which the network is trained for 1000 epochs, each re-training step in the certification loop is done only for a single epoch before verifying the property again. This is because each network already represents fairly accurately the properties we need to certify, usually deviating only in small parts of the total input space.

We define a certification attempt as all the steps required to complete one CEGIS workflow loop, in particular an attempt includes one epoch of training, formal verification of the property, and eventually a counterexample repair. For the size of the networks and properties in this case study, each certification attempt is completed in a few seconds.

When working with multiple properties at the same time, each property is independently verified and its counterexample repaired. Then, all the new training points sampled from each

repaired counterexample are added to the training set before training the network again. This is feasible because the properties are independent from each other.

This approach can also be easily implemented in parallel to further improve the total certification time. However, since we are working on 40 smaller neural networks, we decided to parallelize the entire certification loop for each network instead.

3.4 Results

We certified the eight properties for all relevant networks. In certain instances, the property requires a specific previous advisory; thus, the only certified networks are the ones that share that particular previous advisory.

For each property, we show a table with the accuracy of each network before and after the certification process, together with the number of attempts before the network was certified. In this context, an attempt corresponds to a complete loop in the CEGIS workflow.

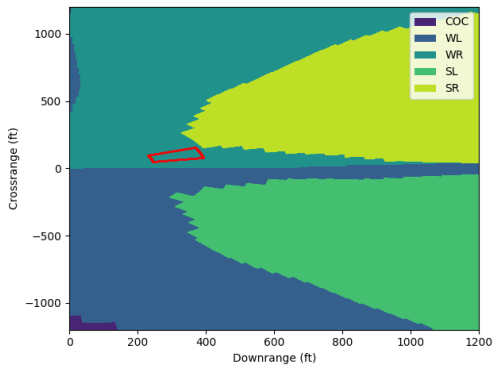
In addition, we visualize the behavior of one of the relevant networks before and after certification. By fixing the relative heading of the intruder to a head-on collision ($\psi = 180°$), we can plot the remaining two variables range (ρ) and heading (θ) in a two-dimensional graph, where for each point in the graph we can observe the advisory chosen by the network. To enhance readability, the polar coordinates are transformed into Cartesian coordinates for plotting purposes. The ownship is always shown in the coordinate (0,0) heading right. The figures show only a subsection of the space around the ownship.

3.4.1 Property 1

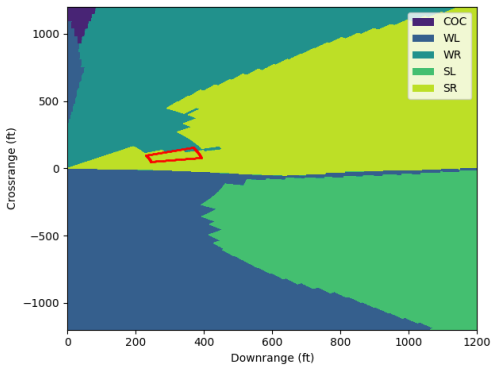
If the intruder is near and approaching from the left, the network advises ***strong right***

All 40 networks can be certified for property φ_1 in an average of 11.1 attempts. Even after the certification process, the accuracy of the networks remains high, with an average reduction of only 0.0087 percentage points.

Figure 3.5 shows the difference in behavior for network $\tau = 0, s_{adv} = COC$ after certification. The space outlined in red corresponds to the area defined by the precondition of φ_1. In this case, we require that every point within the region returns *SR* as the advisory for the property to be satisfied, while before certification it returned *WR*. Due to the limited sample points in the figures, the visualization of the edges between the advisories can be imprecise. However, the networks themselves are formally verified to always satisfy the property.

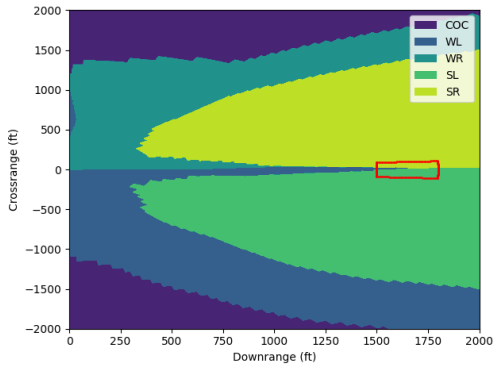


(a) area of interest of φ_1 for network $\tau = 0, s_{adv} = COC$

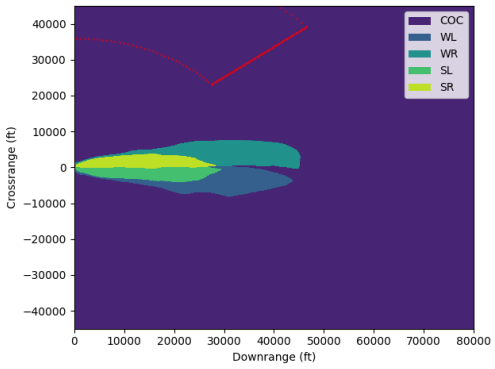


(b) area of interest of φ_1 for network $\tau = 0, s_{adv} = COC$ after certification

Figure 3.5: Change of behavior for the network $\tau = 0, s_{adv} = COC$ after the certification process for property φ_1



(a) area of interest of φ_2



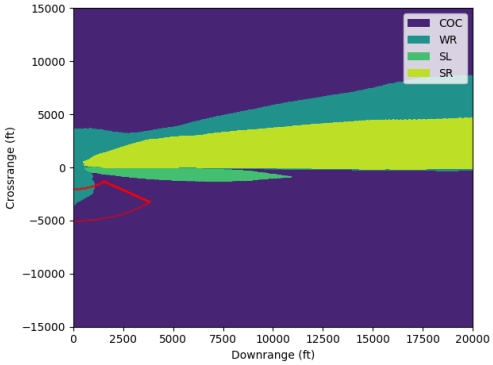
(b) area of interest of φ_3

Figure 3.6: area of interest for the properties φ_2 and φ_3 for network $\tau = 0, s_{adv} = COC$, in this case satisfied without retraining

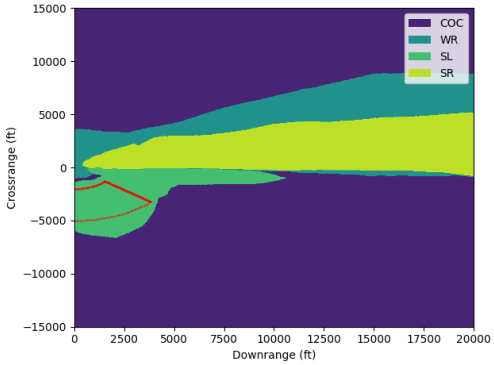
3.4.2 Property 2

If the intruder is directly ahead and is moving towards the ownship, the score for COC will not be maximal

In this instance, it has been verified that the 40 networks already satisfy the property. Consequently, there is no need to repair counterexamples and retrain the networks. For reference, we show the area of interest of the property in Figure 3.6a for the network with $\tau = 0, s_{adv} = COC$



(a) area of interest of φ_4 in network $\tau = 0, s_{adv} = WR$ before certification



(b) area of interest of φ_4 in network $\tau = 0, s_{adv} = WR$ after certification

Figure 3.7: Change of behavior for the network $\tau = 0, s_{adv} = WR$ after the certification process for property φ_4

3.4.3 Property 3

For a far away intruder (on the left), the network advises ***COC***

The property is already satisfied for most neural networks, except in some cases where the previous advisory was *WR* or *SR*. The repaired networks required on average 3.29 attempts and lost 0.0103 percentage points in accuracy.

3.4.4 Property 4

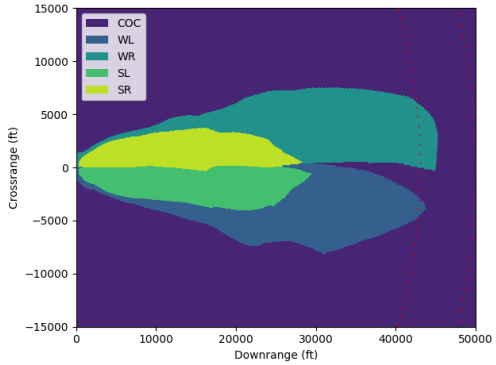
Even if the previous advisory was ***weak right****, the presence of a nearby intruder will cause the network to output a* ***strong left*** *advisory instead*

This property deviates slightly from the original intended behavior of the network, as shown in the comparison Figure 3.7. However, the certification process manages to synthesize the neural networks even if the property is not in agreement with the starting training set, although with significantly more attempts than other properties.

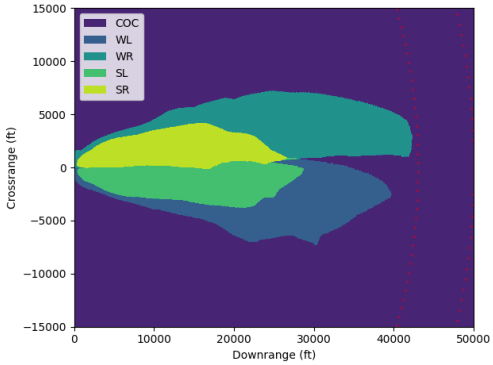
The repaired networks required on average 44.38 attempts and lost 0.0202 percentage points in accuracy.

3.4.5 Property 5

For a far away intruder, the network advises ***COC*** *even if it is in front of the ownship.*



(a) area of interest of φ_5 in network $\tau = 0, s_{adv} = COC$ before certification



(b) area of interest of φ_5 in network $\tau = 0, s_{adv} = COC$ after certification

Figure 3.8: Change of behavior for the network $\tau = 0, s_{adv} = COC$ after the certification process for property φ_5

The repaired networks required on average 6.7 attempts and lost 0.0118 percentage points in accuracy.

3.4.6 Property 6

*If the intruder is near and approaching from the right, the network advises **strong left***

This is the symmetric property of φ_1 on the right side of the ownship instead. The repaired networks required on average 13.0 attempts and lost 0.0043 percentage points in accuracy.

3.4.7 Property 7

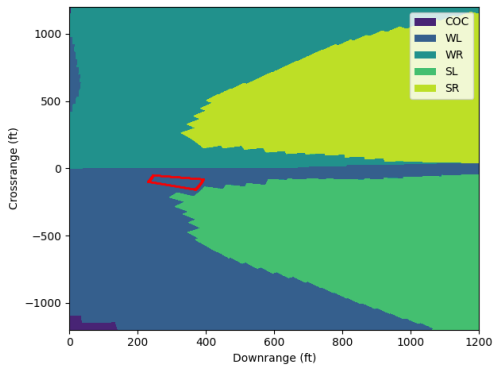
For a far away intruder (on the right), the network advises COC

The symmetric property of φ_2. The property is already satisfied for most neural networks, except in some cases where the previous advisory was *WL* or *SL*.

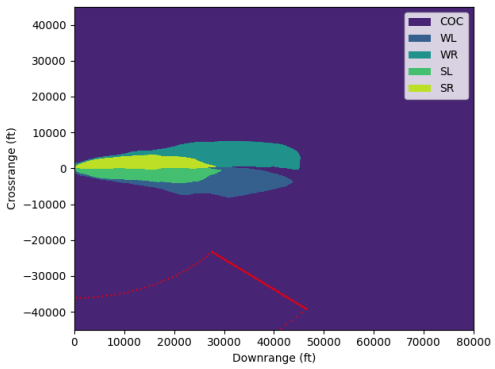
The repaired networks required on average 2.27 attempts and lost 0.0070 percentage points in accuracy.

3.4.8 Property 8

*Even if the previous advisory was **weak left**, the presence of a nearby intruder will cause the network to output a **strong right** advisory instead*

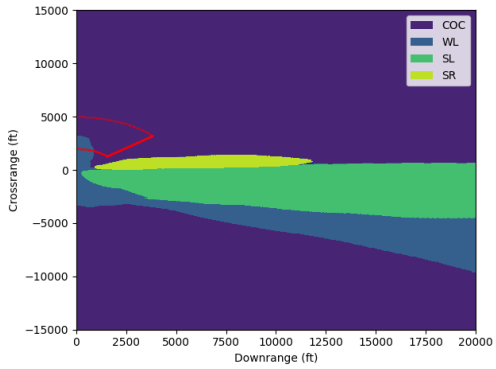


(a) area of interest of φ_6

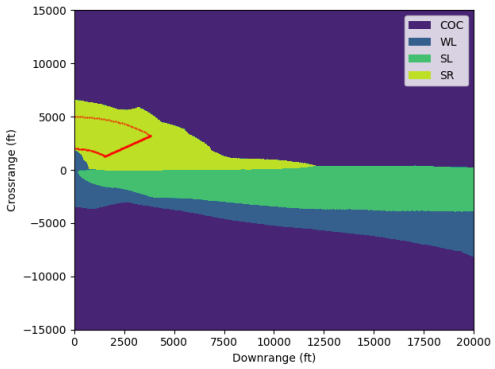


(b) area of interest of φ_7

Figure 3.9: area of interest for the properties φ_6 and φ_7 for network $\tau = 0, s_{adv} = COC$, in this case satisfied without retraining



(a) area of interest of φ_8 in network $\tau = 0, s_{adv} = WL$ before certification



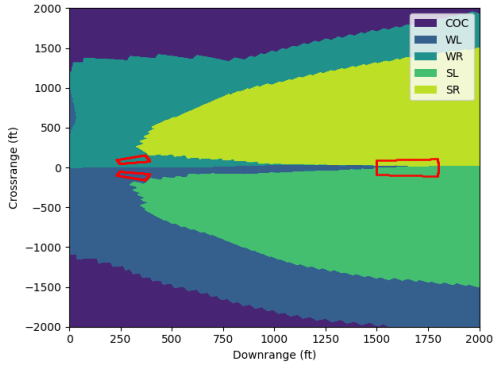
(b) area of interest of φ_8 in network $\tau = 0, s_{adv} = WL$ after certification

Figure 3.10: Change of behavior for the network $\tau = 0, s_{adv} = WL$ after the certification process for property φ_8

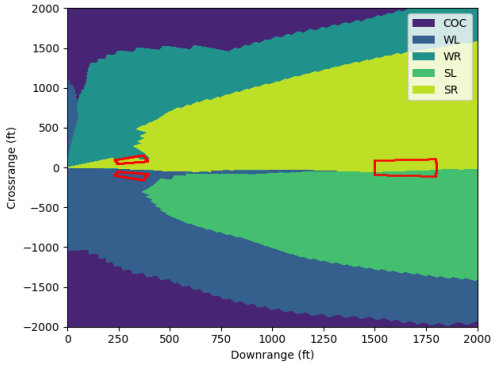
φ_8 is the symmetric version of property φ_4 for the left side of ownship. This property also deviates slightly from the original training set, hence the average number of attempts is higher than other properties. The repaired networks required on average 45.13 attempts and lost 0.0131 percentage points in accuracy.

3.4.9 Multiple properties

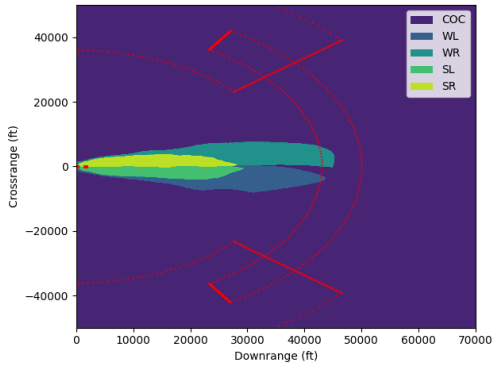
Finally, we test the certification process on multiple properties at the same time. Excluding properties φ_4 and φ_8 that involve only a subset of the networks, the remaining properties φ_1,φ_2 ,φ_3 ,φ_5 ,φ_6 ,φ_7 can be certified for all 40 networks at the same time.



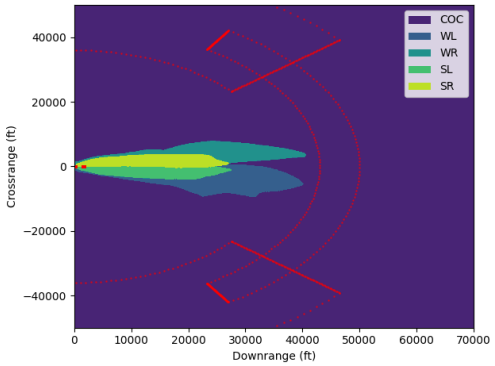
(a) area of interest of $\varphi_1, \varphi_2, \varphi_5$, in network $\tau = 0, s_{adv} = COC$ before certification



(b) area of interest of $\varphi_1, \varphi_2, \varphi_5$, in network $\tau = 0, s_{adv} = COC$ after certification



(c) area of interest of $\varphi_3, \varphi_6, \varphi_7$ in network $\tau = 0, s_{adv} = COC$ before certification



(d) area of interest of $\varphi_3, \varphi_6, \varphi_7$ in network $\tau = 0, s_{adv} = COC$ after certification

Figure 3.11: Change of behavior for the network $\tau = 0, s_{adv} = COC$ after the certification process for property multiple properties

The repaired networks required on average 13.93 attempts and lost 0.0129 percentage points in accuracy. It is crucial to observe that the average number of attempts increases only slightly compared to only one of the properties, such as φ_1 that was certified with 11.1 average attempts. This is due to the design of the CEGIS workflow for multiple properties outlined in Figure 2.2, where all properties are certified, and their repaired counterexamples added to the training set before retraining the network. Thus, the number of training steps remains relatively similar to the hardest single property to certify.

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
COC	00	0.9845	0.9747	-0.0098	3
COC	05	0.9850	0.9763	-0.0087	4
COC	10	0.9878	0.9823	-0.0056	9
COC	15	0.9913	0.9841	-0.0072	5
COC	20	0.9932	0.9888	-0.0045	7
COC	30	0.9951	0.9825	-0.0126	4
COC	40	0.9941	0.9824	-0.0118	9
COC	60	0.9954	0.9920	-0.0034	8
WL	00	0.9902	0.9846	-0.0056	9
WL	05	0.9907	0.9803	-0.0105	19
WL	10	0.9908	0.9846	-0.0062	8
WL	15	0.9927	0.9874	-0.0053	10
WL	20	0.9931	0.9835	-0.0096	27
WL	30	0.9948	0.9903	-0.0045	1
WL	40	0.9395	0.9351	-0.0045	57
WL	60	0.9797	0.9797	0.0000	0
WR	00	0.9779	0.9638	-0.0141	6
WR	05	0.9883	0.9745	-0.0138	13
WR	10	0.9362	0.9321	-0.0041	1
WR	15	0.9896	0.9736	-0.0161	3
WR	20	0.9920	0.9859	-0.0061	16
WR	30	0.9914	0.9789	-0.0124	4
WR	40	0.9956	0.9892	-0.0064	15
WR	60	0.9969	0.9848	-0.0121	20
SL	00	0.9872	0.9824	-0.0048	14
SL	05	0.9883	0.9786	-0.0097	11
SL	10	0.9901	0.9768	-0.0133	9
SL	15	0.9918	0.9776	-0.0142	6
SL	20	0.9943	0.9889	-0.0054	50
SL	30	0.9956	0.9849	-0.0107	12
SL	40	0.9962	0.9877	-0.0085	11
SL	60	0.9963	0.9909	-0.0054	13
SR	00	0.9897	0.9835	-0.0062	2
SR	05	0.9898	0.9733	-0.0165	7
SR	10	0.9920	0.9868	-0.0052	16
SR	15	0.9930	0.9793	-0.0137	11
SR	20	0.9946	0.9827	-0.0119	11
SR	30	0.9959	0.9791	-0.0167	5
SR	40	0.9974	0.9899	-0.0075	2
SR	60	0.9979	0.9911	-0.0068	6

Table 3.3: Accuracy before and after the certification process for φ_1, and certification attempts for all 40 networks

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
COC	00	0.9845	0.9845	0.0	0
COC	05	0.9850	0.9850	0.0	0
COC	10	0.9878	0.9878	0.0	0
COC	15	0.9913	0.9913	0.0	0
COC	20	0.9932	0.9932	0.0	0
COC	30	0.9951	0.9951	0.0	0
COC	40	0.9941	0.9941	0.0	0
COC	60	0.9954	0.9954	0.0	0
WL	00	0.9902	0.9902	0.0	0
WL	05	0.9907	0.9907	0.0	0
WL	10	0.9908	0.9908	0.0	0
WL	15	0.9927	0.9927	0.0	0
WL	20	0.9931	0.9931	0.0	0
WL	30	0.9948	0.9948	0.0	0
WL	40	0.9395	0.9395	0.0	0
WL	60	0.9797	0.9797	0.0	0
WR	00	0.9779	0.9746	-0.0033	5
WR	05	0.9883	0.9829	-0.0054	6
WR	10	0.9362	0.8737	-0.0625	11
WR	15	0.9896	0.9834	-0.0062	2
WR	20	0.9920	0.9867	-0.0052	4
WR	30	0.9914	0.9806	-0.0108	5
WR	40	0.9956	0.9930	-0.0026	3
WR	60	0.9969	0.9895	-0.0074	2
SL	00	0.9872	0.9872	0.0	0
SL	05	0.9883	0.9883	0.0	0
SL	10	0.9901	0.9901	0.0	0
SL	15	0.9918	0.9918	0.0	0
SL	20	0.9943	0.9943	0.0	0
SL	30	0.9956	0.9956	0.0	0
SL	40	0.9962	0.9962	0.0	0
SL	60	0.9963	0.9963	0.0	0
SR	00	0.9897	0.9820	-0.0077	2
SR	05	0.9898	0.9819	-0.0080	1
SR	10	0.9920	0.9920	0.0	0
SR	15	0.9930	0.9861	-0.0069	1
SR	20	0.9946	0.9833	-0.0114	1
SR	30	0.9959	0.9935	-0.0024	1
SR	40	0.9974	0.9925	-0.0049	2
SR	60	0.9979	0.9979	0.0	0

Table 3.4: Accuracy before and after the certification process for φ_3, and certification attempts for all 40 networks

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
WR	00	0.9779	0.9613	-0.0166	32
WR	05	0.9883	0.9665	-0.0218	32
WR	10	0.9362	0.8740	-0.0623	79
WR	15	0.9896	0.9773	-0.0123	29
WR	20	0.9920	0.9817	-0.0102	38
WR	30	0.9914	0.9785	-0.0128	26
WR	40	0.9956	0.9855	-0.0101	64
WR	60	0.9969	0.9817	-0.0152	55

Table 3.5: Accuracy before and after the certification process for φ_4, and certification attempts for networks with $s_{adv} = WR$

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
COC	00	0.9845	0.9783	-0.0062	1
COC	05	0.9850	0.9751	-0.0099	1
COC	10	0.9878	0.9751	-0.0127	2
COC	15	0.9913	0.9864	-0.0049	2
COC	20	0.9932	0.9848	-0.0085	2
COC	30	0.9951	0.9873	-0.0078	5
COC	40	0.9941	0.9919	-0.0023	5
COC	60	0.9954	0.9906	-0.0048	5
WL	00	0.9902	0.9837	-0.0065	8
WL	05	0.9907	0.9778	-0.0130	6
WL	10	0.9908	0.9813	-0.0095	7
WL	15	0.9927	0.9752	-0.0175	7
WL	20	0.9931	0.9807	-0.0124	8
WL	30	0.9948	0.9763	-0.0184	5
WL	40	0.9395	0.9341	-0.0055	10
WL	60	0.9797	0.9703	-0.0094	8
WR	00	0.9779	0.9694	-0.0085	12
WR	05	0.9883	0.9806	-0.0077	16
WR	10	0.9362	0.8455	-0.0907	7
WR	15	0.9896	0.9772	-0.0124	12
WR	20	0.9920	0.9856	-0.0064	7
WR	30	0.9914	0.9655	-0.0258	10
WR	40	0.9956	0.9903	-0.0053	4
WR	60	0.9969	0.9813	-0.0156	10
SL	00	0.9872	0.9801	-0.0072	10
SL	05	0.9883	0.9810	-0.0074	5
SL	10	0.9901	0.9701	-0.0200	7
SL	15	0.9918	0.9785	-0.0132	7
SL	20	0.9943	0.9759	-0.0183	6
SL	30	0.9956	0.9854	-0.0102	5
SL	40	0.9962	0.9891	-0.0071	4
SL	60	0.9963	0.9865	-0.0098	21
SR	00	0.9897	0.9843	-0.0055	5
SR	05	0.9898	0.9816	-0.0082	6
SR	10	0.9920	0.9864	-0.0056	4
SR	15	0.9930	0.9873	-0.0057	4
SR	20	0.9946	0.9847	-0.0100	5
SR	30	0.9959	0.9873	-0.0086	5
SR	40	0.9974	0.9886	-0.0088	6
SR	60	0.9979	0.9939	-0.0040	8

Table 3.6: Accuracy before and after the certification process for φ_5, and certification attempts for all 40 networks

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
COC	00	0.9845	0.9845	0.0000	0
COC	05	0.9850	0.9850	0.0000	0
COC	10	0.9878	0.9878	0.0000	0
COC	15	0.9913	0.9913	0.0000	0
COC	20	0.9932	0.9932	0.0000	0
COC	30	0.9951	0.9951	0.0000	0
COC	40	0.9941	0.9941	0.0000	0
COC	60	0.9954	0.9954	0.0000	0
WL	00	0.9902	0.9902	0.0000	0
WL	05	0.9907	0.9907	0.0000	0
WL	10	0.9908	0.9908	0.0000	0
WL	15	0.9927	0.9927	0.0000	0
WL	20	0.9931	0.9931	0.0000	0
WL	30	0.9948	0.9869	-0.0078	5
WL	40	0.9395	0.9341	-0.0054	27
WL	60	0.9797	0.9797	0.0000	0
WR	00	0.9779	0.9779	0.0000	0
WR	05	0.9883	0.9883	0.0000	0
WR	10	0.9362	0.9362	0.0000	0
WR	15	0.9896	0.9896	0.0000	0
WR	20	0.9920	0.9920	0.0000	0
WR	30	0.9914	0.9914	0.0000	0
WR	40	0.9956	0.9934	-0.0022	1
WR	60	0.9969	0.9927	-0.0042	29
SL	00	0.9872	0.9872	0.0000	0
SL	05	0.9883	0.9883	0.0000	0
SL	10	0.9901	0.9901	0.0000	0
SL	15	0.9918	0.9918	0.0000	0
SL	20	0.9943	0.9943	0.0000	0
SL	30	0.9956	0.9956	0.0000	0
SL	40	0.9962	0.9962	0.0000	0
SL	60	0.9963	0.9943	-0.0020	3
SR	00	0.9897	0.9897	0.0000	0
SR	05	0.9898	0.9898	0.0000	0
SR	10	0.9920	0.9920	0.0000	0
SR	15	0.9930	0.9930	0.0000	0
SR	20	0.9946	0.9946	0.0000	0
SR	30	0.9959	0.9959	0.0000	0
SR	40	0.9974	0.9974	0.0000	0
SR	60	0.9979	0.9979	0.0000	0

Table 3.7: Accuracy before and after the certification process for φ_6, and certification attempts for all 40 networks

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
COC	00	0.9845	0.9845	0.0000	0
COC	05	0.9850	0.9850	0.0000	0
COC	10	0.9878	0.9878	0.0000	0
COC	15	0.9913	0.9913	0.0000	0
COC	20	0.9932	0.9932	0.0000	0
COC	30	0.9951	0.9951	0.0000	0
COC	40	0.9941	0.9941	0.0000	0
COC	60	0.9954	0.9954	0.0000	0
WL	00	0.9902	0.9848	-0.0055	2
WL	05	0.9907	0.9841	-0.0066	5
WL	10	0.9908	0.9832	-0.0077	2
WL	15	0.9927	0.9884	-0.0042	3
WL	20	0.9931	0.9771	-0.0160	2
WL	30	0.9948	0.9848	-0.0100	1
WL	40	0.9395	0.9347	-0.0049	5
WL	60	0.9797	0.9695	-0.0102	3
WR	00	0.9779	0.9779	0.0000	0
WR	05	0.9883	0.9883	0.0000	0
WR	10	0.9362	0.9362	0.0000	0
WR	15	0.9896	0.9896	0.0000	0
WR	20	0.9920	0.9920	0.0000	0
WR	30	0.9914	0.9914	0.0000	0
WR	40	0.9956	0.9956	0.0000	0
WR	60	0.9969	0.9969	0.0000	0
SL	00	0.9872	0.9801	-0.0072	2
SL	05	0.9883	0.9883	0.0000	0
SL	10	0.9901	0.9865	-0.0036	1
SL	15	0.9918	0.9831	-0.0087	1
SL	20	0.9943	0.9860	-0.0083	2
SL	30	0.9956	0.9885	-0.0070	2
SL	40	0.9962	0.9927	-0.0035	2
SL	60	0.9963	0.9952	-0.0011	1
SR	00	0.9897	0.9897	0.0000	0
SR	05	0.9898	0.9898	0.0000	0
SR	10	0.9920	0.9920	0.0000	0
SR	15	0.9930	0.9930	0.0000	0
SR	20	0.9946	0.9946	0.0000	0
SR	30	0.9959	0.9959	0.0000	0
SR	40	0.9974	0.9974	0.0000	0
SR	60	0.9979	0.9979	0.0000	0

Table 3.8: Accuracy before and after the certification process for φ_7, and certification attempts for all 40 networks

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
WL	00	0.9902	0.9788	-0.0115	34
WL	05	0.9907	0.9829	-0.0078	30
WL	10	0.9908	0.9782	-0.0126	18
WL	15	0.9927	0.9860	-0.0067	41
WL	20	0.9931	0.9669	-0.0262	45
WL	30	0.9948	0.9799	-0.0149	25
WL	40	0.9395	0.9277	-0.0118	109
WL	60	0.9797	0.9665	-0.0132	59

Table 3.9: Accuracy before and after the certification process for φ_8, and certification attempts for networks where $s_{adv} = WL$

s_{adv}	τ	Accuracy Before	Accuracy After	Difference	Attempts
COC	00	0.9845	0.9821	-0.0024	6
COC	05	0.9850	0.9741	-0.0109	5
COC	10	0.9878	0.9679	-0.0199	9
COC	15	0.9913	0.9806	-0.0107	6
COC	20	0.9932	0.9854	-0.0079	7
COC	30	0.9951	0.9898	-0.0053	7
COC	40	0.9941	0.9788	-0.0154	12
COC	60	0.9954	0.9864	-0.0091	9
WL	00	0.9902	0.9836	-0.0066	9
WL	05	0.9907	0.9813	-0.0094	19
WL	10	0.9908	0.9734	-0.0175	10
WL	15	0.9927	0.9802	-0.0125	10
WL	20	0.9931	0.9781	-0.0150	49
WL	30	0.9948	0.9140	-0.0808	38
WL	40	0.9395	0.9358	-0.0037	6
WL	60	0.9797	0.9742	-0.0055	7
WR	00	0.9779	0.9640	-0.0139	8
WR	05	0.9883	0.9777	-0.0105	14
WR	10	0.9362	0.9233	-0.0129	16
WR	15	0.9896	0.9775	-0.0121	24
WR	20	0.9920	0.9706	-0.0214	17
WR	30	0.9914	0.9798	-0.0115	12
WR	40	0.9956	0.9831	-0.0125	23
WR	60	0.9969	0.9883	-0.0086	15
SL	00	0.9872	0.9781	-0.0091	16
SL	05	0.9883	0.9809	-0.0075	12
SL	10	0.9901	0.9749	-0.0152	13
SL	15	0.9918	0.9814	-0.0103	9
SL	20	0.9943	0.9814	-0.0128	41
SL	30	0.9956	0.9842	-0.0113	10
SL	40	0.9962	0.9854	-0.0107	11
SL	60	0.9963	0.9814	-0.0149	36
SR	00	0.9897	0.9808	-0.0090	4
SR	05	0.9898	0.9785	-0.0113	6
SR	10	0.9920	0.9798	-0.0123	18
SR	15	0.9930	0.9861	-0.0069	12
SR	20	0.9946	0.9787	-0.0160	11
SR	30	0.9959	0.9815	-0.0144	7
SR	40	0.9974	0.9914	-0.0060	5
SR	60	0.9979	0.9849	-0.0130	8

Table 3.10: Accuracy before and after the certification process for φ_1, φ_2, φ_3, φ_5, φ_6, φ_7 and certification attempts for all 40 networks

4 Conclusion

In this thesis, we demonstrate an automatic synthesis method for certified neural networks based on the CEGIS workflow. Starting from the formal verification of a property, we obtain counterexamples when the property is not satisfied. Then, we exploit these counterexamples to generate new correct training points using a novel approach based on Quadratic Programming. Subsequently, we retrain the network on the original dataset augmented with the new training points, until no further counterexamples are found.

We show a practical example of automatic synthesis on the ACAS Xu dataset, where we successfully synthesize 40 different neural networks to be certified for 8 safety properties while still maintaining high accuracy of classification. Due to the modular nature of the CEGIS loop, we can incorporate existing state-of-the-art solvers into the implementation, improving the speed and efficiency of the process.

Future work may explore different neural network architectures, for example, convolutional neural networks or unsupervised learning techniques. Alternative strategies could involve larger neural networks and more complex properties where the scalability and efficiency of the solution become essential.

References

[1] S. Pertigkiozoglou and P. Maragos, "Detecting adversarial examples in convolutional neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1812.03303

[2] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 2014. [Online]. Available: https://arxiv.org/abs/1312.6199

[3] G. Katz, C. Barrett, D. Dill, K. Julian, and M. Kochenderfer, "Reluplex: An efficient smt solver for verifying deep neural networks," 2017. [Online]. Available: https://arxiv.org/abs/1702.01135

[4] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić, D. L. Dill, M. J. Kochenderfer, and C. Barrett, "The marabou framework for verification and analysis of deep neural networks," in *Computer Aided Verification*, I. Dillig and S. Tasiran, Eds. Cham: Springer International Publishing, 2019, pp. 443–452.

[5] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," 2018. [Online]. Available: https://arxiv.org/abs/1804.10829

[6] S. Srivastava, S. Gulwani, and J. S. Foster, "From program verification to program synthesis," *SIGPLAN Not.*, vol. 45, no. 1, p. 313–326, Jan. 2010. [Online]. Available: https://doi.org/10.1145/1707801.1706337

[7] S. Gulwani, S. Jha, A. Tiwari, and R. Venkatesan, "Synthesis of loop-free programs," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 62–73. [Online]. Available: https://doi.org/10.1145/1993498.1993506

[8] S. Gulwani, W. R. Harris, and R. Singh, "Spreadsheet data manipulation using examples," *Commun. ACM*, vol. 55, no. 8, p. 97–105, Aug. 2012. [Online]. Available: https://doi.org/10.1145/2240236.2240260

[9] R. Alur, R. Bodik, G. Juniwal, M. M. K. Martin, M. Raghothaman, S. A. Seshia, R. Singh, A. Solar-Lezama, E. Torlak, and A. Udupa, "Syntax-guided synthesis," in *2013 Formal Methods in Computer-Aided Design*, 2013, pp. 1–8.

[10] R. Alur, D. Fisman, R. Singh, and A. Solar-Lezama, "Sygus-comp 2016: results and analysis," *Electronic Proceedings in Theoretical Computer Science*, vol. 229, pp. 178–202, 2016.

[11] A. Solar-Lezama, L. Tancau, R. Bodik, S. Seshia, and V. Saraswat, "Combinatorial sketching for finite programs," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 5, p. 404–415, Oct. 2006. [Online]. Available: https://doi.org/10.1145/1168917.1168907

[12] F. Bauer-Marquart, D. Boetius, S. Leue, and C. Schilling, *SpecRepair: Counter-Example Guided Safety Repair of Deep Neural Networks*. Springer International Publishing, 2022, p. 79–96. [Online]. Available: http://dx.doi.org/10.1007/978-3-031-15077-7_5

[13] D. Boetius, S. Leue, and T. Sutter, "A robust optimisation perspective on counterexample-guided repair of neural networks," 2023. [Online]. Available: https://arxiv.org/abs/2301.11342

[14] A. Albarghouthi, "Introduction to neural network verification," 2021. [Online]. Available: https://arxiv.org/abs/2109.10317

[15] M. J. Kochenderfer, C. Amato, G. Chowdhary, J. P. How, H. J. D. Reynolds, J. R. Thornton, P. A. Torres-Carrasquillo, N. K. Ure, and J. Vian, *Optimized Airborne Collision Avoidance*, 2015, pp. 249–276.

[16] M. Marston and G. Baca, *ACAS-Xu initial self-separation flight tests*. NASA, DFRC-E-DAA-TN22968, 2015.

[17] K. D. Julian, M. J. Kochenderfer, and M. P. Owen, "Deep neural network compression for aircraft collision avoidance systems," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 3, p. 598–608, Mar. 2019. [Online]. Available: http://dx.doi.org/10.2514/1.G003724

[18] K. D. Julian and M. J. Kochenderfer, "Guaranteeing safety for neural network-based aircraft collision avoidance systems," in *2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)*. IEEE, Sep. 2019, p. 1–10. [Online]. Available: http://dx.doi.org/10.1109/DASC43569.2019.9081748

[19] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, *Efficient BackProp*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 9–48. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_3

[20] J. Bai, F. Lu, K. Zhang *et al.*, "Onnx: Open neural network exchange," https://github.com/onnx/onnx, 2019.

[21] H. Wu, O. Isac, A. Zeljić, T. Tagomori, M. Daggitt, W. Kokke, I. Refaeli, G. Amir, K. Julian, S. Bassan, P. Huang, O. Lahav, M. Wu, M. Zhang, E. Komendantskaya, G. Katz, and C. Barrett, "Marabou 2.0: A versatile formal analyzer of neural networks," 2024. [Online]. Available: https://arxiv.org/abs/2401.14461

[22] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2024. [Online]. Available: https://www.gurobi.com