Exact Preimages of Neural Network Aircraft Collision Avoidance Systems

Kyle Matoba
Idiap and École Polytechnique Fédérale de Lausanne
kyle.matoba@epfl.ch

François Fleuret*
University of Geneva
francois.fleuret@unige.ch

Abstract

A common pattern of progress in engineering has seen deep neural networks displacing human-designed logic. There are many advantages to this approach, but divorcing decisionmaking from human oversight and intuition has costs as well. One is that deep neural networks can map similar inputs to very different outputs in a way that makes their application to safety-critical domains problematic.

We present a method to check that the decisions of a deep neural network are as intended by constructing the exact preimage of its predictions. Preimages generalize verification in the sense that they can be used to verify a wide class of properties, and answer much richer questions besides. We examine the functioning of an aircraft collision avoidance system, and show how exact preimages reduce undue conservatism when examining dynamic safety.

Our method iterates backwards through the layers of piecewise linear deep neural networks. Uniquely, we compute *all* intermediate values that correspond to a prediction, propagating this calculation through layers using analytical formulae for layer preimages.

1 Introduction

A recognition that deep neural network (DNN)-based image classifiers can deliver very different predictions for visually indistinguishable inputs (Szegedy et al. [11]) has highlighted the need for an understanding of the worst-case behavior of DNNs. We are interested in this problem in the engineering context, where compared to image classifiers, DNNs tend to be smaller, simpler, and to map from smaller domains to smaller ranges. Additionally, both input and output values tend to have definite interpretation, and the phenomenon being modelled is governed by understood physical principles. In such problems, human intuition can improve verification, and should be incorporated, since overwhelmingly such DNNs form a component for a larger pipeline in which inaccuracies can propagate. These considerations are exemplified by DNN-based aircraft automated collision avoidance systems (ACAS).

ACAS are navigational aids that use data on positions and velocities to issue guidance on evasive actions to prevent collisions with an intruding aircraft. The ACAS developed in Kochenderfer and Chryssanthacopoulos [9] uses dynamic programming to formulate the optimal control of a partially observed Markov process, and issues advisories to optimize a criterion that penalizes near collisions and raising false or inconsistent warnings. Unfortunately, evaluating the policy function is too resource-intensive to run on certified avionics hardware. Small DNNs have been been found to be adequate approximators that require little storage and can perform inference quickly. A downside of this approximation is that even accurate DNNs can give very wrong predictions on some inputs – Katz et al. [8], for example show that when another aircraft is nearby and approaching from the left, a DNN-based approximation need not advise the correct action of turning right aggressively.

*Work done at Idiap

Workshop on machine learning for engineering modeling, simulation and design @ NeurIPS 2020

Commonly called	What is computed	Examples
Verification	$(f, X, Y) \mapsto \mathbf{1}_{f^{-1}(Y)\cap X=\emptyset} (= \mathbf{1}_{f(X)\cap Y=\emptyset})$	Wong and Kolter [13]
Reachability	$(f, X) \mapsto f(X)$	Yang et al. [14]
Inversion	$(f, y) \mapsto f^{-1}(\{y\})$	Carlsson et al. [2]
Preimage	$(f, Y) \mapsto f^{-1}(Y)$	This paper

Table 1: A taxonomy of previous work on inversion and verification. Here $f : \mathbb{R}^{n_1} \to \mathbb{R}^{n_L}$ is a DNN, $X \subseteq \mathbb{R}^{n_1}$, $x \in \mathbb{R}^{n_1}$, $Y \subseteq \mathbb{R}^{n_L}$, and $y \in \mathbb{R}^{n_L}$. f^{-1} is its inverse in the sense that $f^{-1}(Y) = \{x : f(x) \in Y\}$.

1.1 Previous work

Table 1 orients our work to the literature.

Verification amounts to a simple yes or no, and so answering higher-level questions typically requires many verifications: for example, Katz et al. [8] describes a suite of 45 tests, and an image classifier might wish to verify the absence of adversarial examples around the entire training set. Yang et al. [14] is an interesting extension to verification in that it computes the entire image of, say, an epsilon ball around a data point, and not just whether it intersects with a decision boundary.

Reasoning about the outputs that can arise from inputs is only half of the picture. Carlsson et al. [2] and Behrmann et al. [1] attempt to reconstruct the inputs that result in an output, studying the statistical invariances that nonlinear layers encode. Behrmann et al. [1] examines the preimage of a single point through a single ReLU layer, assessing stability via an approximation-based experiment. Carlsson et al. [2] analyzes the preimage of a single point through the repeated application of a nonlinearity, purely theoretically. Our paper looks at the preimage of non-singleton subsets of the codomain, which is important in the engineering context, and requires considerable extension to their approaches.

2 Dynamic properties of DNN-based ACAS

Verification can check that one-step behavior in a DNN-based ACAS behaves as intended. However, it cannot answer higher level questions like "will a near-collision occur if this policy is followed?" The idea of Julian and Kochenderfer [5] is to verify dynamic properties of such systems by combining single-step verification with worst-case assumptions about randomness in state transitions and (constrained) behavior of other aircraft.

In Julian and Kochenderfer [5], the state consists of x and y distances between the two aircraft, and an angle of approach angle between them, ψ. The actions are five turning advisories: (1) "clear of conflict" (COC), (2) weak left [turn] (WL), (3) strong left (SL), (4) weak right (WR), and (5) strong right (SR). The initial condition is given by the boundary of the domain where the distance of the intruding aircraft are at their maxima. Transition dynamics are denoted by $\Psi(a, S)$, a set-valued function which gives the set of states that are reachable from states in S under action a. Ψ encompasses both randomness in the transition, and behavior of the other aircraft. The change in (x, y) is controlled by the angle between the crafts, and the update to the angle is the difference between the turning of the two crafts, with some randomness. To compute the states that can arise under a policy, the idea is to begin from an initial set of states that are known to be reachable, and to iteratively append states that are reachable from any of those states, until a fixed point is reached. U denotes the set of states that we wish to preclude.

3 Discretize and verify: Julian and Kochenderfer [5]

This idea is formalized by Julian and Kochenderfer [5] as Algorithm 1. Because multiple advisories will be issued whenever a cell straddles the decision boundary, the discretized algorithm will wrongly include some states as reachable since a worst-case analysis needs to take account of *all* reachable states. Table 2 gives an indication of the magnitudes of overestimation, presenting how much of the state space will lead to multiple advisories under a simple discretization scheme.

Julian and Kochenderfer [5] do not use an equispaced grid, but the basic point – that discretization error cannot be made negligible – is an inescapable feature of this approach. And any false positives in a single-step decision function will be amplified in the dynamic analysis, as more reachable states at one point time lead to even more reachable points at the next step, so a 1% overestimation at one step may be compounded to considerably more through the dynamics. Coincidentlly, Julian and Kochenderfer [5] are able to reach a usable solution, but cannot guarantee the absence of near collisions under some realistic parameter configuations.

g	Volume fraction
40	0.05128
80	0.02532
120	0.01681
160	0.01267
200	0.01005

Table 2: Each dimensions of (x, y, ψ) is discretized into a grid of size g. We present the fraction of the g^3 cubes for which all eight corners do not evaluate to the same prediction, which is a sufficient condition for the cell to straddle a decision boundary.

Data: Maximum distance set $\mathcal{R}_0$, policy f, an "unsafe set" U, transition dynamics Ψ, encounter length T.
Result: Guaranteed to not reach an unsafe state from $\mathcal{R}_0$ under policy f?
initialization: $t = 0$, done = False;
Partition the state space into cells $c \in \mathcal{C}$;
while *not done* **do**
  $t = t + 1$;
  $\mathcal{R}_t = \emptyset$;
  for $c \in \mathcal{C}$ *such that* $c \cap \mathcal{R}_{t-1} \neq \emptyset$ **do**
    for i *such that* $f(c) \cap \{x : x_i \geq x_j \text{ for } j \neq i\} \neq \emptyset$ **do**
      for $c' \in \mathcal{C}$ *such that* $c' \cap \Psi(i, c) \neq \emptyset$ **do**
        $\mathcal{R}_t \leftarrow \mathcal{R}_t \cup c'$
      end
    end
  end
  done = $\mathcal{R}_t == \mathcal{R}_{t-1}$ or $U \cap \mathcal{R}_t \neq \emptyset$ or $t > T$.
end
Return $\mathcal{R}_t \cap U == \emptyset$

Algorithm 1: Algorithm from Julian and Kochenderfer [5] for computing whether an unsafe set U can be reached under a policy f beginning from $\mathcal{R}_0$ under transition dynamics Ψ.

4 Our preimage-based alternative

Note how the cells in Algorithm 1 can be traversed in any order. This is a simple way to see that this algorithm is not fully utilizing the spatial structure of the problem. Rather than looping first over the domain and secondly over actions at those points, Algorithm 2 incorporates a knowledge of the DNN behavior by looping over actions and using the preimage to simultaneously compute all reachable points under that action.

To action this algorithm, we need to calculate $f^{-1}(\{x : x_i \geq x_j \text{ for } j \neq i\})$, the preimage of all five turning advisories. First, we note that the preimage of the composition of functions is the reversed composition of preimages.

Lemma 1. *For functions* $f_j : \mathbb{R}^{n_j} \to \mathbb{R}^{n_{j+1}}$,

$$(f_{\ell+k} \circ f_{\ell+k-1} \circ \ldots \circ f_\ell)^{-1} = f_\ell^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_{\ell+k}^{-1}. \tag{1}$$

This is an elementary property of invertible functions, but holds more generally for essentially the same reason. Secondly, we mention an intuitive property of f^{-1} that is handy for building up the preimage of any set from the preimages of any partition of that set.

Lemma 2 (Preimage of union is union of preimages)**.**

$$f^{-1}\left(\cup_{i=1}^N S_i\right) = \cup_{i=1}^N f^{-1}(S_i).$$

Our method starts from a set of the form $\{x \in \mathbb{R}^n : b - Ax \geq 0\}$ for some $m \in \mathbb{N}$, $b \in \mathbb{R}^m$, and $A \in \mathbb{R}^{m \times n}$, called a polytope. We examine layers such that $f_\ell^{-1}(\{x : b - Ax \geq 0\})$ is of the form

Data: $\mathcal{R}_0$, f, U, Ψ, T.
Result: Guaranteed to not reach an unsafe state from $\mathcal{R}_0$ under policy f?
initialization: $t = 0$, done = False;
for $i = 1, 2, \ldots, n_L$ **do**
  $\Xi_i = f^{-1}(\{x : x_i \geq x_j \text{ for } j \neq i\})$
end
while *not done* **do**
  $t = t + 1$;
  $\mathcal{R}_t = \emptyset$;
  for $i = 1, 2, \ldots, n_L$ **do**
    $\mathcal{R}_t \leftarrow \mathcal{R}_t \cup \Psi(i, \Xi_i \cap \mathcal{R}_{t-1})$;
  end
  done = $\mathcal{R}_t == \mathcal{R}_{t-1}$ or $U \cap \mathcal{R}_t \neq \emptyset$ or $t > T$.
end
Return $U \cap \mathcal{R}_t == \emptyset$.

Algorithm 2: Our preimage-based, exact algorithm for computing the dynamically reachable states in an ACAS.

$\bigcup_{i=1}^{N} \{x : b_i - A_i x \geq 0\}$ for $N \in \mathbb{N}, b_i \in \mathbb{R}^{m_i}$, $A_i \in \mathbb{R}^{m_i \times n}$ – meaning that the preimage of f_ℓ is a union of polytopes. By Lemma 2, the preimage of such sets can be computed by computing the preimage of each term in the union: in short the problem is self-similar, though increasing numbers of the problem need be solved with each iteration. Clearly $\{x : x_i \geq x_j \text{ for } j \neq i\}$ is a polytope, so all that remains is to demonstrate that f is the composition of such functions.

Lemma 3 (Preimage of Linear layer)**.**

$$(x \mapsto Wx + a)^{-1}(\{x : b - Ax \geq 0\}) = \{x : (b - Aa) - AWx \geq 0\}. \tag{2}$$

ReLU is a piecewise linear function, so if we carefully treat the portions of the domain on which it exhibits different behavior, we obtain a similar formulation for each:

Lemma 4 (Preimage of ReLU layer)**.**

$$\begin{aligned} &\text{ReLU}^{-1}(\{x : b - Ax \geq 0\}) \\ &= \bigcup_{\nu \in \{0,1\}^n} \{x : b - A\text{diag}(\nu)x \geq 0, -\text{diag}(1-\nu)x \geq 0, \text{diag}(\nu)x \geq 0\}. \end{aligned} \tag{3}$$

To understand Lemma 4 let $s(x)$ be the vector given by $s(x)_i = 1$ if $x_i \geq 0$ and zero otherwise. Then $\text{diag}(s(x))x = \text{ReLU}(x)$. This expression separates $x \mapsto \text{ReLU}(x)$ into a pattern of signs over its coordinates and x itself. This means that once we restrict attention to a set on which the sign does not change, we can apply familiar linear algebra routines to compute the preimage set, akin to Lemma 3. The nonnegative values are denoted by $\nu \in \{0,1\}^n$ in the above, and the set of x such that $x_i \geq 0 \iff \nu_i = 1$ is given by $\text{diag}(\nu)x \geq 0$. Similarly, $x_i \leq 0 \iff \nu_i = 0$ for $i = 1, 2, \ldots, n$ if and only if $-\text{diag}(1-\nu)x \geq 0$. Equation 3 follows by partitioning $\mathbb{R}^n$ into the 2^n sets where each coordinate is nonnegative or not.

We have only phrased the argument for DNNs comprised of linear and ReLU layers, those employed in Katz et al. [8], Julian et al. [6], and Julian and Kochenderfer [5], but essentially the same argument applies for a very wide class of DNNs employed across application areas, including image classifiers. This is discussed in Appendix B.

5 Analysis

While Algorithm 2 is exact – it will never wrongly say that a state can be reached – the accuracy of Algorithm 1 is ultimately controlled by the number of cells, $|\mathcal{C}|$. This is because it is necessary to perform n_L verifications for each reachable cell, and the number of reachable cells is proportional to $|\mathcal{C}|$. Let V denote the cost of a verification. Verification is known to be NP-complete (Katz et al. [8]), so V dominates all others calculation such as computing intersections or evaluating $\Psi(i, c)$. Thus, the computational cost of Algorithm 1 is $O(|\mathcal{C}|Vn_L)$. In Algorithm 2 must initially compute

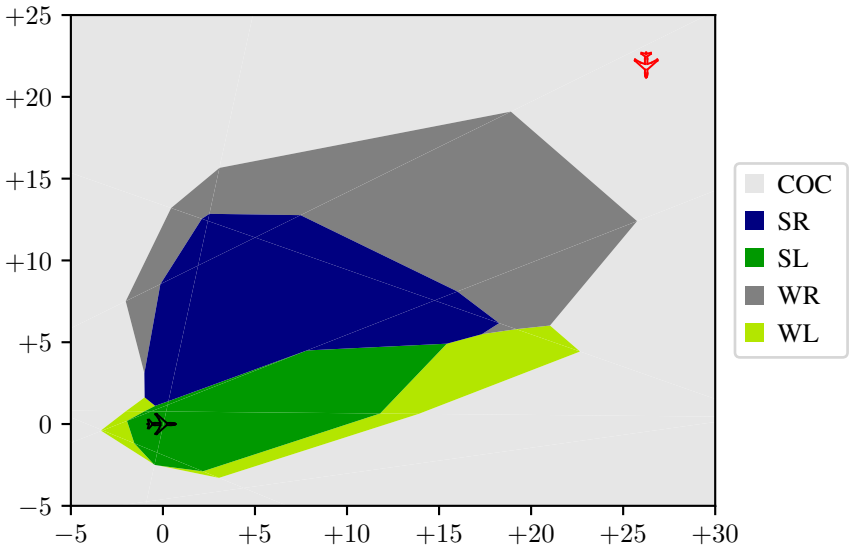

Figure 1: An encounter plot showing the optimal action at each (x, y) distance configuration for a fixed angle of approach, indicated by the perpendicular orientation of the red (intruder) aircraft. Distances are measured in kilofeet.

n_L preimages which dominates the entire calculation, which consists of relatively fast operations – applying the dynamics and computing intersections up to T times, for T a number around 40.

Let P denote the cost of computing a preimage, then Algorithm 2 is $O(Pn_L)$. So whilst it dispenses with the need to solve $O(|\mathcal{C}|)$ verifications, it may nonetheless be more intractable if P is significantly greater than V. Let the dimensions of the nonlinear layers in a DNN be n_{ℓ_i}, then because in the worst case it is necessary to check each nonlinearity, each of which can be independently in a negative or positive configuration, $V = O(2^{\sum_i n_{\ell_i}})$. *Exact verification for even a single cell is impossible at present for large networks.* We believe that preimages can be computed roughly as easily as a verification – $P = O(V)$. We are currently developing this conjecture formally, the idea is that, as shown in Lemma 4, each nonlinear layer ℓ_i generates up to $2^{n_{\ell_i}}$ sets, the preimage of which must be computed through earlier layers.

In any case, as is true of any exponentially hard problem, the practical tractability of both P and V hinges importantly upon theoretical arguments showing that not all 2^n configurations of the nonlinearities of an n-dimensional layer can be achieved (Serra et al. [10] and Hanin and Rolnick [3], and clever implementations that take account of the structure of the problems (e.g. Tjeng et al. [12] and Katz et al. [8]).

The distinction between the two algorithms is made clearer by examining an *encounter plot* such as Figure 1. Encounter plots are concise summarizations of the policy function, here depicting the possible advisories, for a fixed angle of approach (conveyed by the orientation of the red aircraft relative to the black). This figure, which replicates Figure 4 of Julian and Kochenderfer [5], differs from it in a crucial respect: it depicts the *analytically-computed* preimage of the five sets where each of the advisories are issued (details of the experiment are in Appendix D). The shaded areas arise from plotting polytopes, as in Algorithm 2. Julian and Kochenderfer [5], on the other hand, produce such plots by evaluating the predictions of the network on a fine grid. The different manner in which the plots are produced is an exact analogue of the different way that the networks are summarized and analyzed through time.

6 Conclusion

In this paper, we analyzed a DNN-based ACAS via its *preimage* – computing the set of inputs that map to each of its outputs. We showed how a the preimages of a partition of the range of a DNN can

be better suited to certain types of analysis, for example computing the set of states that an aircraft following an ACAS can reach over time. In this case, we saw that it can eliminate uncertainty around the true decision boundaries, and may also be faster to compute if the preimage can be computed quickly enough. Work currently in progress addresses seeks to address this question.

7 Acknowledgements

This work has benefitted from the feedback of Suraj Srinivas, Martin Jaggi, and Pascal Frossard. Kyle Matoba was supported by the Swiss National Science Foundation under grant number FNS-188758 "CORTI".

References

[1] Jens Behrmann et al. "Analysis of Invariance and Robustness via Invertibility of ReLU-Networks". In: *arXiv e-prints* (2018).

[2] Stefan Carlsson, Hossein Azizpour, and Ali Sharif Razavian. "The Preimage of Rectifier Network Activities". In: 2017.

[3] Boris Hanin and David Rolnick. "Deep ReLU Networks Have Surprisingly Few Activation Patterns". In: *arXiv e-prints* (2019).

[4] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[5] Kyle D. Julian and Mykel J. Kochenderfer. "Guaranteeing safety for neural network-based aircraft collision avoidance systems". In: *IEEE/AIAA 38th Digital Avionics Systems Conference (DASC)* (2019).

[6] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. "Deep Neural Network Compression for Aircraft Collision Avoidance Systems". In: *Journal of Guidance, Control, and Dynamics* 42.3 (2019), pp. 598–608.

[7] Kyle Julian et al. "Policy compression for aircraft collision avoidance systems". In: *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*. Sept. 2016, pp. 1–10.

[8] Guy Katz et al. "Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks". In: *Computer Aided Verification*. Ed. by Rupak Majumdar and Viktor Kunčak. Springer International Publishing, 2017.

[9] Mykel J. Kochenderfer and James P. Chryssanthacopoulos. "Robust Airborne Collision Avoidance through Dynamic Programming". In: *Massachusetts Institute of Technology, Lincoln Laboratory, Project Report ATC-371* (2011).

[10] Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. "Bounding and Counting Linear Regions of Deep Neural Networks". In: *arXiv e-prints* (2017).

[11] Christian Szegedy et al. "Intriguing properties of neural networks". In: (2014). 2nd International Conference on Learning Representations, ICLR 2014 ; Conference date: 14-04-2014 Through 16-04-2014.

[12] Vincent Tjeng, Kai Xiao, and Russ Tedrake. "Evaluating Robustness of Neural Networks with Mixed Integer Programming". In: *arXiv e-prints* (2017).

[13] Eric Wong and Zico Kolter. "Provable defenses against adversarial examples via the convex outer adversarial polytope". In: *arXiv e-prints* (2017).

[14] Xiaodong Yang et al. "Reachability Analysis for Feed-Forward Neural Networks using Face Lattices". In: *arXiv e-prints*, arXiv:2003.01226 (2020), arXiv:2003.01226.

A Proofs

A.1 Proof of Lemma 1

Proof. Unroll Equation 1. Let $S \subseteq \mathbb{R}^{n_{\ell+k}}$ be arbitrary.

$$\begin{aligned}
&(f_{\ell+k} \circ f_{\ell+k-1} \circ \ldots \circ f_\ell)^{-1}(S) \\
=&\{x : (f_{\ell+k} \circ f_{\ell+k-1} \circ \ldots \circ f_\ell)(x) \in S\} \\
=&\{x : f_{\ell+k}((f_{\ell+k-1} \circ f_{\ell+k-2} \circ \ldots \circ f_\ell)(x)) \in S\} \\
=&\{x : (f_{\ell+k-1} \circ f_{\ell+k-2} \circ \ldots \circ f_\ell)(x) \in f_{\ell+k}^{-1}(S)\} \\
=&\vdots \\
=&\{x : (f_{\ell+1} \circ f_\ell)(x) \in (f_{\ell+2}^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_{\ell+k}^{-1})(S)\} \\
=&\{x : f_\ell(x) \in (f_{\ell+1}^{-1} \circ f_{\ell+2}^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_{\ell+k}^{-1})(S)\} \\
=&(f_\ell^{-1} \circ \ldots \circ f_{\ell+k-1}^{-1} \circ f_\ell + k)^{-1}(S).
\end{aligned} \tag{4}$$

□

A.2 Proof of Lemma 2

Proof.

$$\begin{aligned}
&x \in f^{-1}(\cup_{i=1}^N S_i) \iff \\
&f(x) \in \cup_{i=1}^N S_i \iff \\
&f(x) \in S_1 \text{ or } f(x) \in S_2 \text{ or } \ldots \text{ or } f(x) \in S_N \iff \\
&x \in f^{-1}(S_1) \text{ or } x \in f^{-1}(S_2) \in S_2 \text{ or } \ldots \text{ or } x \in f^{-1}(S_N) \iff \\
&x \in \cup_{i=1}^N f^{-1}(S_i).
\end{aligned}$$

□

Note that an identical argument shows that $f^{-1}\left(\cap_{i=1}^N S_i\right) = \cap_{i=1}^N f^{-1}(S_i)$. This can be useful in some applications where where S_i can be wrtten as $\Psi \cap \Xi_i$ – writing $\cup_i S_i$ as $\Psi \cap \cup_i \Xi_i$ may be more efficient.

B Inverting general DNN classifiers

In familiar terms a DNN classifier might consist of some "feature building" modules, say composed of alternating convolution and maxpooling, then flattened, and passed onto the prediction logic consisting of alternating linear and ReLU layers, possibly including dropout or batch normalization, and concluding with a softmax function to normalize the predictions to a probability distribution.

How do the results of Section 4 suffice to invert such DNNs? Firstly, under our convention that layers operate on flat tensors, flattening is superfluous. Next, dropout affects inference only through the weights – this layer can be omitted entirely in computing the preimage. Convolution is essentially linear. Maxpool is straightforwardly rewritten in terms of the ReLU and linear function. $\{x : b - A\text{softmax}(x) \geq 0\}$ is not a polytope. However, if the classification alone suffices then the softmax layer can be elided entirely since $\arg\max_j \ x_j = \arg\max_j \ \text{softmax}(x)_j$.

Resnets (He et al. [4]) do not strictly fit this pattern, but can be handled with similar reasoning. The key function in a residual block is

$$x \mapsto W_2 \text{ReLU}(W_1 x) + x.$$

Combining arguments similar to Lemma 3 and Lemma 4, we have

Lemma 5 (Preimage of residual block)**.**

$$\begin{aligned}
&(z \mapsto W_2\text{ReLU}(W_1 z) + z)^{-1}(\{x : b - Ax \geq 0\}) \\
=&\{x : b - A(W_2\text{ReLU}(W_1 x) + x) \geq 0\} \\
=&\bigcup_{\nu \in \{0,1\}^n} \{x : b - A(W_2\text{diag}(\nu)W_1 + I)x \geq 0, -\text{diag}(1-\nu)W_1 x \geq 0, \text{diag}(\nu)W_1 x \geq 0\}\,.
\end{aligned} \tag{5}$$

C Collecting this all up

Section 4 described a method for computing the preimage of polytope through a DNN that can be written as the composition of linear and ReLU functions. Appendix B argued that the extension to a broader class of networks is straightforward. For completeness, we here give a more detailed summary of the steps:

1. Put the network into "standard form":
   (a) Embed any transformations that are "off" at inference time, such as dropout or batch normalization into the weights.
   (b) Rewrite the network in flattened form, for example replacing $3 \times 32 \times 32$ tensors by 3072×1 vectors. This is a convention to facilitate our polytope formulation.
   (c) Rewrite all transformations as compositions of linear and ReLU functions. For example, convolution and average pooling are linear functions. Maxpooling, hard tanh, and leaky ReLU can be written as the composition of linear and ReLU functions.
2. Let $f = f_L \circ f_{L-1} \circ \ldots \circ f_1$ denote the network in this form.
3. Let $R_L = \cup_i \Delta_i$ be the image set that we wish to invert, for example $R_L = \Delta_1 = \{x : x_1 \geq x_2\} \subseteq \mathbb{R}^2$ in a binary classifier.
4. Compute $f_L^{-1}(\Delta_i)$ for all i, using Lemma 3, Lemma 4, Lemma 5, or other analytical formulae.
5. Each term above is a union of polytopes, thus $\cup_i f_L^{-1}(\Delta_i)$ is a union of polytopes.
6. By Lemma 2, $R_{L-1} \triangleq f_L^{-1}(R_L) = \cup_i f_L^{-1}(\Delta_i)$.
7. R_{L-1} is *also* a union of polytopes, so apply the same argument to compute $R_{L-2} \triangleq f_{L-1}^{-1}(R_{L-1}) = f_{L-1}^{-1}(f_L^{-1}(R_L))$.
8. Repeat for $\ell = L-2, \ldots, 1$ to compute $R_0 = f_1^{-1}(R_1) = \ldots = (f_1^{-1} \circ f_2^{-1} \circ \ldots \circ f_L^{-1})(R_L)$.
9. Appeal to Equation 1 to conclude that $R_0 = f^{-1}(R_L)$.

D Further computation details

The analysis presented in Section 5 is based entirely on data generated by Julian and Kochenderfer [5]'s system that formulates and solves dynamic programs to deliver lookup tables of optimal collision avoidance behavior in the same manner as the FAA's proprietary software.[2] Our DNN modelling is somewhat different, however, and whilst we think that our results can be interpreted within their framework, in this section we detail the aspects of our analysis that differ from Julian and Kochenderfer [5]'s.

D.1 Fitting – Optimization Criterion

The first manner in which our approach is different is the fitting criterion: Julian and Kochenderfer [5] issue advisories as a function of position and velocities indirectly: by first fitting the continuation value to taking each action, and then choosing the action with the highest predicted continuation value. This oblique approach is understandable: this work is the continuation of an extended project

[2] `https://github.com/sisl/HorizontalCAS`

to build (Kochenderfer and Chryssanthacopoulos [9]) and compress the Q-table (Julian et al. [7], Katz et al. [8]).

And although the Q-values themselves have some interpretation, issuing advisories requires only knowing the greatest. We hypothesize that it is easier to solve a problem which recognizes an invariance of the prediction to any increasing transformation. And that is what we find – by replacing Julian et al. [7]'s mean-squared-error-based criterion with cross entropy loss to directly model the optimal decision, we are able to achieve better performance with smaller networks. One statement of the improvement is that Julian and Kochenderfer [5] use a five layer fully connected layers with 25 neurons each to achieve an accuracy of 97% to 98%. We are able to achieve comparable accuracy with a *two* layer 25 neuron fully connected network (a network of the same size targetting MSE loss only attains an accuracy around 93%).

Why is anything less than complete fidelity to the Q-table acceptable in an approximation? The answer seems to be twofold: firstly the Q-table is itself not perfect, because of discretization artifacts. One can observe physically implausible sawtooth-like decision boundaries that arise from a coarse grid in the top plot of Julian and Kochenderfer [5] Figure 4. The second is that accuracy alone does not capture the genuine metric of goodness, for example in the bottom plot of Figure 4 of Julian and Kochenderfer [5] we see a highly accurate network that exhibits unusual "islands" of SR completely encompassed by a region of WR that are both not present in the ground truth, and also prescibe a conceptually wrong relationship (a pilot could be initially advised a strong right turn, then after some period of lessened danger have it downgraded to a weak right, only to have it re-upgraded to a strong right, seemingly although the danger continues to lessen). The correct metric seems to rather be plausibility of the prescribed encounter plot. These observation leads us to not worry too much about small differences in model accuracies in favour of plausibility of the encounter plots.

D.2 Fitting – Symmetry

The second manner in which our approach differs from Julian and Kochenderfer [5] is in the domain being fitted. Julian and Kochenderfer [5] fixed a lookup table over $(x, y, \psi) \in [-56000, +56000]^2 \times [-\pi, +\pi)$. However, if we let $Q : \mathbb{R}^3 \to \mathbb{R}^5$ denote the Q-function as a function of the state $s = (x, y, \psi)$, then the physics of the problem insure that

$$Q(T_i s) = T_o Q(s) \text{ where } T_i = \begin{pmatrix} +1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{pmatrix} \text{ and } T_o = \begin{pmatrix} +1 & 0 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & +1 \\ 0 & 0 & 0 & +1 & 0 \end{pmatrix}.$$

This relationship clearly only works for $a_{\text{prev}} = \text{COC}$, but similar symmetries will exist more generally. Thus, strictly speaking, half of the lookup table is unneeded, and moreover it would seem wasteful to ask a network to learn (what we already know to be) the same thing twice. Thus, our method is to only fit f over $(x, y, \psi) \in [-56000, +56000]^2 \times [0, +\pi)$, and when needed to infer $f(s) = T_o f(T_i s)$ for $s = (x, y, \psi)$ with $\psi < 0$. In so doing, we halve the data set size, but leave other data fitting parameter unchanged.

To continue the analysis above describing comparable performance from smaller networks, exploiting symmetry enables us to achieve accuracy above 97% from a one layer, 24 neuron network. For computational ease, Figure 1 is computed on a 16 neuron network that achieves about 96% accuracy.

D.3 Inversion – Projection

Figure 1 was formed by taking the fitted $n_0 = 3$ DNN, fixing ψ to a given value, and working with the resultant $n_0 = 2$ DNN – if W_1, b_1 denote the weights and bias of the original DNN, we invert the network having weight and bias $W_1' = W_1 \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{pmatrix}, b_1' = b_1 + \psi W_1 \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.