Safe Predictors for Input-Output Specification Enforcement

Abstract

This paper presents an approach for designing neural networks, along with other machine learning models, which adhere to a collection of input-output specifications. Our method involves the construction of a constrained predictor for each set of compatible constraints, and combining these predictors in a safe manner using a convex combination of their predictions. We demonstrate the applicability of this method with synthetic datasets and on an aircraft collision avoidance problem.

1 Introduction

The increasing adoption of machine learning models, such as neural networks, in safety-critical applications, such as autonomous vehicles and aircraft collision avoidance, highlights an urgent need for the development of guarantees on safety and robustness. These models may be required to satisfy specific input-output specifications to ensure the algorithms comply with physical laws, can be executed safely, and are consistent with prior domain knowledge. Furthermore, these models should demonstrate adversarial robustness, meaning their outputs should not change abruptly within small input regions – a property that neural networks often fail to satisfy.

Recent studies have shown the capacity to verify formally input-output specifications and adversarial robustness properties of neural networks. For instance, the Satisability Modulo Theory (SMT) solver Reluplex was employed to verify properties of networks being used in the Next-Generation Aircraft Collision Avoidance System for Unmanned aircraft (ACAS Xu). Reluplex has also been used to verify adversarial robustness. While Reluplex and other similar techniques can effectively determine if a network satisfies a given specification, they do not offer a way to guarantee that the network will meet those specifications. Therefore, additional methods are needed to adjust networks if it is found that they are not meeting the desired properties.

There has been an increase in techniques for designing networks with certified adversarial robustness, but enforcing more general safety properties in neural networks is still largely unexplored. One approach to achieving provably correct neural networks is through abstraction-refinement optimization. This approach has been applied to the ACAS-Xu dataset, but the network was not guaranteed to meet the specifications until after training. Our work seeks to design networks with enforced input-output constraints even before training has been completed. This will allow for online learning scenarios where a system has to guarantee safety throughout its operation.

This paper presents an approach for designing a safe predictor (a neural network or any other machine learning model) that will always meet a set of constraints on the input-output relationship. This assumes that the constrained output regions can be formulated to be convex. Our correct-by-construction safe predictor is guaranteed to satisfy the constraints, even before training, and at every training step. We describe our approach in Section 2, and show its use in an aircraft collision avoidance problem in Section 3. Results on synthetic datasets can be found in Appendix B.

.

2 Method

Considering two normed vector spaces, an input space X and an output space Y , and a collection of c different pairs of input-output constraints, (A_i, B_i), where $A_i \subseteq X$ and B_i is a convex subset of Y for each constraint i, the goal is to design a safe predictor, $F : X \to Y$, that guarantees $x \in A_i \Rightarrow F(x) \in B_i$.

Let b be a bit-string of length c. Define O_b as the set of points z such that, for all i, $b_i = 1$ implies $z \in A_i$, and $b_i = 0$ implies $z \notin A_i$. O_b thus represents the overlap regions for each combination of input constraints. For example, O_{101} is the set of points in A_1 and A_3, but not in A_2, and $O_{0\ldots0}$ is the set where no input constraints apply. We also define O as the set of bit strings, b, such that O_b is non-empty, and define $k = |O|$. The sets $\{O_b : b \in O\}$ create a partition of X according to the combination of input constraints that apply.

Given:

- c different input constraint proximity functions, $\sigma_i : X \to [0, 1]$, where σ_i is continuous and $\forall x \in A_i$, $\sigma_i(x) = 0$,
- k different constrained predictors, $G_b : X \to B_b$, one for each $b \in O$, such that the domain of each G_b is non-empty,

We define:

- a set of weighting functions, $w_b(x) = \frac{\prod_{i:b_i=1}(1-\sigma_i(x))\prod_{i:b_i=0}\sigma_i(x)}{\sum_{b\in O}\prod_{i:b_i=1}(1-\sigma_i(x))\prod_{i:b_i=0}\sigma_i(x)}$, where $\sum_{b\in O} w_b(x) = 1$, and
- a safe predictor, $F(x) = \sum_{b\in O} w_b(x)G_b(x)$.

Theorem 2.1. For all i, if $x \in A_i$, then $F(x) \in B_i$.

A formal proof of Theorem 2.1 is presented in Appendix A and can be summarized as: if an input is in A_i, then by construction of the proximity and weighting functions, all of the constrained predictors, G_b, that do not map to B_i will be given zero weight. Only the constrained predictors that map to B_i will be given non-zero weight, and because of the convexity of B_i, the weighted average of the predictions will remain in B_i.

If all G_b are continuous and if there are no two input sets, A_i and A_j, for which $(A_i \cap A_j) \subset (\partial A_i \cup \partial A_j)$, then F will be continuous. In the worst case, as the number of constraints grows linearly, the number of constrained predictors needed to describe our safe predictor grows exponentially. In practice, however, we expect many of the constraint overlap sets, O_b, to be empty. Consequently, any predictors corresponding to an empty set can be ignored. This significantly reduces the number of constrained predictors needed for many applications.

See Figure 1 for an illustrative example of how to construct $F(x)$ for a notional problem with two overlapping input-output constraints.

2.1 Proximity Functions

The proximity functions, σ_i, describe how close an input, x, is to a particular input constraint region, A_i. These functions are used to compute the weights of the constrained predictors. A desirable property for σ_i is for $\sigma_i(x) \to 1$ as $d(x, A_i) \to \infty$, for some distance function. This ensures that when an input is far from a constraint region, that constraint has little influence on the prediction for that input. A natural choice for such a function is:

$$\sigma_i(x; \Sigma_i) = 1 - \exp\left(-\frac{d(x, A_i)}{\sigma_1}\right)^{\sigma_2}.$$

Here, Σ_i is a set of parameters $\sigma_1 \in (0, \infty)$ and $\sigma_2 \in (1, \infty)$, which can be specified based on engineering judgment, or learned using optimization over training data. In our experiments in this paper, we use proximity functions of this form and learn independent parameters for each input-constrained region. We plan to explore other choices for proximity functions in future work.

2.2 Learning

If we have families of differentiable functions $G_b(x;\theta_b)$, continuously parameterized by θ_b, and families of $\sigma_i(x;\chi_i)$, differentiable and continuously parameterized by χ_i, then $F(x;\Theta,X)$, where $\Theta = \{\theta_b : b \in O\}$ and $X = \{\chi_i : i = 1, ..., c\}$, is also continuously parameterized and differentiable. We can thus apply standard optimization techniques (e.g., gradient descent) to find parameters of F that minimize a loss function on some dataset, while also preserving the desired safety properties. Note that the safety guarantee holds regardless of the parameters. To create each $G_b(x;\theta_b)$ we consider choosing:

- a latent space $\mathbb{R}^m$,
- a map $h_b : \mathbb{R}^m \to B_b$,
- a standard neural network architecture $g_b : X \to \mathbb{R}^m$,

and then defining $G_b(x;\theta_b) = h_b(g_b(x;\theta_b))$.

The framework proposed here does not require an entirely separate network for each b. In many applications, it may be advantageous for the constrained predictors to share earlier layers, thus creating a shared representation of the input space. In addition, our definition of the safe predictor is general and is not limited to neural networks.

In Appendix B, we show examples of applying our approach to synthetic datasets in 2-D and 3-D with simple neural networks. These examples show that our safe predictor can enforce arbitrary input-output specifications using convex output constraints on neural networks, and that the learned function is smooth.

3 Application to Aircraft Collision Avoidance

Aircraft collision avoidance requires robust safety guarantees. The Next-Generation Collision Avoidance System (ACAS X), which issues advisories to prevent near mid-air collisions, has both manned (ACAS Xa) and unmanned (ACAS Xu) variants. The system was originally designed to choose optimal advisories while minimizing disruptive alerts by solving a partially observable Markov decision process. The solution took the form of a large look-up table, mapping each possible input combination to scores for all possible advisories. The advisory with the highest score would then be issued. By using a deep neural network (DNN) to compress the policy tables, it has been necessary to verify that the DNNs meet certain safety specifications.

A desirable ž01csafeability ž01d property for ACAS X was defined in a previous work. This property speci01ed that for any given input state within the ž01csafeable region,ž01d an advisory would never be issued that could put the aircraft into a state where a safe advisory would no longer exist. This concept is similar to control invariance. A simplified model of the ACAS Xa system was created, named VerticalCAS. DNNs were then generated to approximate the learned policy, and Reluplex was used to verify whether the DNNs satisfied the safeability property. This work found thousands of counterexamples where the DNNs did not meet the criteria.

Our approach for designing a safe predictor ensures any collision avoidance system will meet the safeability property by construction. Appendix C describes in detail how we apply our approach to a subset of the VerticalCAS datasets using a conservative, convex approximation of the safeability constraints. These constraints are defined such that if an aircraft state is in the "unsafeable region", $A_{\text{unsafeable},i}$, for the i^{th} advisory, the score for that advisory must not be the highest, i.e., $x \in A_{\text{unsafeable},i} \Rightarrow F_i(x) < \max_j F_j(x)$, where $F_j(x)$ is the output score for the j^{th} advisory.

Table 1 shows the performance of a standard, unconstrained network and our safe predictor. For both networks, we present the percentage accuracy (ACC) and violations (percentage of inputs for which the network outputs an unsafe advisory). We train and test using PyTorch with two separate datasets, based on the previous advisory being Clear of Conflict (COC) and Climb at 1500 ft/min (CL1500). As shown in the table, our safe predictor adheres to the required safeability property. Furthermore, the accuracy of our predictor remains the same as the unconstrained network, demonstrating we are not losing accuracy to achieve safety guarantees.

Table 1: Results of the best configurations of β-TCVAE on DCI, FactorVAE, SAP, MIG, and IRS metrics.

NETWORK	ACC (COC)	VIOLATIONS (COC)	ACC (CL1500)	VIOLATIONS (CL1500)
STANDARD	96.87	0.22	93.89	0.20
SAFE	96.69	0.00	94.78	0.00

4 Discussion and Future Work

We propose an approach for designing a safe predictor that adheres to input-output specifications for use in safety-critical machine learning systems, demonstrating it on an aircraft collision avoidance problem. The novelty of our approach is its simplicity and guaranteed enforcement of specifications through combinations of convex output constraints during all stages of training. Future work includes adapting and using techniques from optimization and control barrier functions, as well as incorporating notions of adversarial robustness into our design, such as extending the work to bound the Lipschitz constant of our networks.

Appendix A: Proof of Theorem 2.1

Proof. Fix i and assume that $x \in A_i$. It follows that $\sigma_i(x) = 0$, so for all $b \in O$ where $b_i = 0$, $w_b(x) = 0$. Thus,

$$F(x) = \sum_{b \in O, b_i = 1} w_b(x) G_b(x).$$

If $b_i = 1$, $G_b(x) \in B_i$, and thus $F(x)$ is also in B_i by the convexity of B_i.

Appendix B: Example on Synthetic Datasets

Figure 2 depicts an example of applying our safe predictor to a notional regression problem. This example uses inputs and outputs in 1-D with one input-output constraint. The unconstrained network consists of a single hidden layer with a dimension of 10, ReLU activations, and a fully connected layer. The safe predictor shares this structure with the unconstrained network but has its own fully connected layer for the constrained predictors, G_0 and G_1. Training uses a sampled subset of points from the input space. Figure 3 shows an example of applying our safe predictor to a notional regression problem with a 2-D input and 1-D output, using two overlapping constraints. The unconstrained network has two hidden layers of dimension 20 and ReLU activations, followed by a fully connected layer. The constrained predictors, G_{00}, G_{10}, G_{01}, and G_{11}, share the hidden layers but also have an additional hidden layer of size 20 with ReLU, followed by a fully connected layer. Training uses a sampled subset of points from the input space.

Appendix C: Details of VerticalCAS Experiment

C.1 Safeability Constraints

The "safeability" property, originally introduced and used to verify the safety of the VerticalCAS neural networks can be encoded into a set of input-output constraints. The "safeable region" for a given advisory represents input locations where that advisory can be selected such that future advisories exist that will prevent an NMAC. If no future advisories exist, the advisory is "unsafeable" and the corresponding input region is the "unsafeable region". Examples of these regions, and their proximity functions are shown in Figure 5 for the CL1500 advisory.

The constraints we enforce are that $x \in A_{\text{unsafeable},i} \Rightarrow F_i(x) < \max_j F_j(x), \forall i$, where $A_{\text{unsafeable},i}$ is the unsafeable region for the i^{th} advisory, and $F_j(x)$ is the output score for the j^{th} advisory. Because the output regions of the safeable constraints are not convex, we make a conservative approximation, enforcing $F_i(x) = \min_j F_j(x)$, for all $x \in A_{\text{unsafeable},i}$.

C.2 Proximity Functions

We start by generating the unsafeable region bounds from the open source code. We then compute a "distance function" between input space points (vO - vI, h, τ), and the unsafeable region for each advisory. These are not true distances but are 0 if and only if the data point is within the unsafeable set. These are then used to produce proximity functions as given in Equation 1.

C.3 Structure of Predictors

The compressed policy tables for ACAS Xu and VerticalCAS use neural networks with six hidden layers with a dimension of 45, and ReLU activation functions. We used the same architecture for the unconstrained network. For our constrained predictors, we use the same structure but have shared first four layers for all predictors. This provides a common learned representation of the input space, while allowing each predictor to adapt to its own constraints. After the shared layers, each constrained predictor has an additional two hidden layers and their final outputs are projected onto our convex approximation of the safe region of the output space, using $G_b(x) = \min_j G_j(x)$. In our experiments, we set $\epsilon = 0.0001$.

With this construction, we needed 30 separate predictors to enforce the VerticalCAS safeability constraints. The number of nodes for the unconstrained and safe implementations were 270 and 2880, respectively. Our safe predictor is orders of magnitude smaller than the original look-up tables.

C.4 Parameter Optimization

We use PyTorch for defining our networks and performing parameter optimization. We optimize both the unconstrained and safe predictors using the asymmetric loss function to select advisories while also accurately predicting scores. The data is split using an 80/20 train/test split with a random seed of 0. The optimizer is ADAM with a learning rate of 0.0003 and batch size of 216, with training for 500 epochs.

Appendix A: Proof of Theorem 2.1

Proof. Let $x \in A_i$. Then, $\sigma_i(x) = 0$, and for all $b \in O$ where $b_i = 0$, $w_b(x) = 0$. Thus,

$$F(x) = \sum_{b \in O, b_i = 1} w_b(x) G_b(x)$$

If $b_i = 1$, then $G_b(x) \in B_i$, and therefore $F(x)$ is in B_i due to the convexity of B_i.

Appendix B: Example on Synthetic Datasets

Figure 2 depicts an example of applying our safe predictor to a notional regression problem with 1-D input and outputs, and one input-output constraint. The unconstrained network has a single hidden layer of dimension 10 with ReLU activations, followed by a fully connected layer. The safe predictor shares this structure with constrained predictors, G_0 and G_1, but each predictor has its own fully connected layer. The training uses a sampled subset of points from the input space and the learned predictors are shown for the continuous input space.

Figure 3 shows an example of applying the safe predictor to a notional regression problem with a 2-D input and 1-D output and two overlapping constraints. The unconstrained network has two hidden layers of dimension 20 with ReLU activations, followed by a fully connected layer. The constrained predictors G_{00}, G_{10}, G_{01} and G_{11} share the hidden layers and have an additional hidden layer of size 20 with ReLU followed by a fully connected layer. Again, training uses a sampled subset of points from the input space and the learned predictors are shown for the continuous input space.

Appendix C: Details of VerticalCAS Experiment

C.1 Safeability Constraints

The "safeability" property from prior work can be encoded into a set of input-output constraints. The "safeable region" for a given advisory is the set of input space locations where that advisory can be chosen, for which future advisories exist that will prevent an NMAC. If no future advisories exist for preventing an NMAC, the advisory is deemed "unsafeable," and the corresponding input region is the "unsafeable region." Figure 5 shows an example of these regions for the CL1500 advisory.

The constraints we enforce in our safe predictor are: $x \in A_{\text{unsafeable},i} \Rightarrow F_i(x) < \max_j F_j(x)$, $\forall i$. To make the output regions convex, we approximate by enforcing $F_i(x) = \min_j F_j(x)$, for all $x \in A_{\text{unsafeable},i}$.

C.2 Proximity Functions

We start by generating the bounds on the unsafeable regions. Then, a distance function is computed between points in the input space ($v_O - v_I$, h, τ), and the unsafeable region for each advisory. While these are not true distances, their values are 0 if and only if the data point is inside the unsafeable set. When used to produce proximity functions as given in Equation 1, these values help ensure safety. Figure 5 shows examples of the unsafeable region, distance function, and proximity function for the CL1500 advisory.

C.3 Structure of Predictors

The compressed versions of the policy tables from prior work are neural networks with six hidden layers, 45 dimensions in each layer, and ReLU activation functions. We use the same architecture for our standard, unconstrained network. For constrained predictors, we use a similar architecture. However, the first four hidden layers are shared between all of the predictors. This learns a single, shared input space representation, and also allows each predictor to adapt to its constraints. Each constrained predictor has two additional hidden layers and their outputs are projected onto our convex approximation of the safe output region. We accomplish this by setting the score for any unsafeable advisory i to $G_i(x) = \min_j G_j(x) - \epsilon$. In our experiments, we used $\epsilon = 0.0001$.

To enforce the VerticalCAS safeability constraints, we need 30 separate predictors. This increases the size of the network from 270 to 2880 nodes for the unconstrained and safe implementations respectively. However, our safe predictor remains smaller than the original look-up tables by several orders of magnitude.

C.4 Parameter Optimization

We define our networks and perform parameter optimization using PyTorch. We optimize the parameters of both the unconstrained network and our safe predictor using the asymmetric loss function, guiding the network to select optimal advisories while accurately predicting scores from the look-up tables. Each dataset is split using an 80/20 train/test split, with a random seed of 0. The optimizer is ADAM, with a learning rate of 0.0003, a batch size of 216, and the number of training epochs is 500.