CIS 7000-005: Debugging Data & Models Fall 2022

Verification — October 13

Prof. Eric Wong

Sometimes, we may want to check our models for certain properties. Most commonly these are safety properties, i.e. checking if a controller behaves in a safe manner.

- What kind of properties do we want to check for machine learning models?
- How do we check for these properties?

1 Safety properties of neural networks

The classic use-case for verification tis to check for safety properties of a neural network. One of the earliest examples of this was used for an Airborne Collision Avoidance System (ACAS Xu). This system advised horizontal maneuvers for aircraft to avoid collisions. The system was originally developed as a giant lookup table that mapped sensor measurements to specific advisories, but this required over 2GB of memory. Apparently, 2GB of memory is a cause for concern for avionics equipment, so a neural network representation was developed instead as a replacement for the lookup table. To make it even faster, this was replaced with 45 different neural networks running in parallel, and the final compressed form uses less than 3MB of memory.

The cost of this highly compressed and optmized neural network representation is its complexity—it is not so straightforward to guarantee that the neural network does not give an erroneous alert. For aircraft, where the risks of error are extraordinarily high, we need to be able to prove that a set of inputs does not produce an error. How do we prove this?

- Previous methodology was to exhaustively test the system in 1.5 million simulated encounters. Does this prove correctness for neural networks? No, because of the continuous nature of the neural network decision space.
- Verification techniques that solve optimization problems to guarantee correctness, or provide a counterexample.

ACAS Xu The input for ACAS Xu is the following:

- ρ distance from ownship to intruder
- θ Angle to intruder relative to ownship heading direction
- ψ Heading angle of intruder relative to ownship
- v_{own}: speed of ownship
- v_{int} speed of intruder

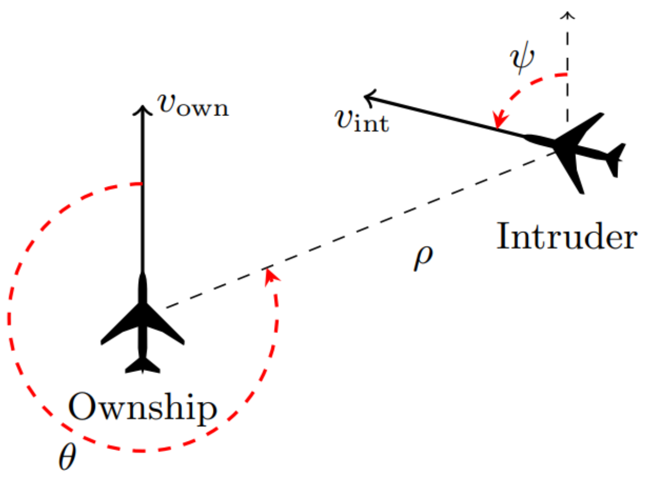

Figure 1: Geometry for Acas Xu Logic table (figure from Katz et al. 2017)

- τ time until loss of vertical separation
- a_{prev} previous advisory

The output of the network are 5 different advisories. These are

- Clear of conflict (CoC)
- Weak right
- Strong right
- Weak left
- Strong left

The last two variables tau, a_{prev} were discretized into 45 different pairs and a separate neural network trained for each one. Thus each neural network has 5 inputs and 5 outputs. The networks are small and simple—6 hidden layers of 300 hidden units each with ReLU activation.

Properties For the aircraft system, we desire to verify several properties. These properties fall in three categories:

- The system does not give unnecessary turning advisories
- Alerting regions are uniform and do not contain inconsistent alerts
- Strong alerts do not appear for high τ

For example, if the intruder is far away the network should advise CoC. If the intruder is nearby and approach from the lft, the network should advise strong right. If vertical separation is large, then the network never advises a strong turn.

To check these properties with an off the shelf SAT, SMT, or MILP solver, 4 hours was not enough. Specialized solvers for neural networks are necessary. Inded, verifying properties (i.e. those with linear constraints on the input and linear outputs) is NP-complete (there is a reduction from 3-SAT).

Adversarial robustness Another property that is commonly checked for with verification tools is adversarial robustness, or the presence of adversarial examples. Here, the input constraint is usually the ℓ_2 or ℓ_∞ ball around an example, and the desired property is that the predicted class remains the maximal score throughout the entire input region.

2 Verification as optimization

All of these properties can be formulated as $c^T f(x) \geq 0$ for all x such that $Ax \leq b$. The checking of this property can then be formulated as the following optimization problem:

$$\min_{Ax \leq b} c^T f(x) \tag{1}$$

where f is an eural network, $Ax \leq b$ is the input constraint, and $c^T f(x)$ is the property being checked. If $c^T f(x^\star) \geq 0$ for the optimal solution $x^\star$, then we can conclude that $c^T f(x) \geq 0$ for all x that satisfy the constraints. But this is generally a non-convex and complicated problem, since f is a neural network.

it is typically more convenient to lift the neural network into the constraints of the optimization problem. Specifically:

$$\begin{aligned} &\min_z c^T z_k \\ \text{subject to } & z_{i+1} = \max(0, \tilde{z}_{i+1}) \\ & \tilde{z}_{i+1} = W_i z_i + b_i \text{ for } i = 0 \dots k-1 \\ & Az_0 \leq b \end{aligned} \tag{2}$$

where the objective and input constraints are the same, but the layers of a neural network are now encoded as constraints within an constrained optimization problem. Here, we can see that this is almost all linear except for the ReLU constraints.

Many verification techniques can be characterized by how they handle the ReLU constraints. All of these techniques rely on having lower and upper bounds $\ell_i \leq \tilde{z}_i \leq u_i$ on the ReLU.

MILP In mixed integer linear programming, we can solve optimization problems with linear constraints and integer constraints. Here, since almost everything linear, the main challenge is to encode ReLU as an mixed integer/linear constraint. The way we do this is as follows (Xiao et al. 2017), where the constraint $y = \max(0, x)$ is equivalent to

$$(y \leq x - \ell(1-a)) \wedge (y \geq x) \wedge (y \leq u \cdot a) \wedge (y \geq 0) \wedge (a \in \{0, 1\}) \tag{3}$$

This integer encoding only needs to be done when the ReLU is unstable, i.e. $\ell \leq 0 \leq u$.

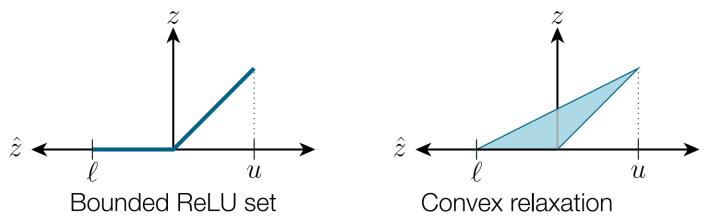

Figure 2: Convex outer bound of the ReLU activation

LP Linear programming is a relaxation of the MILP. While the MILP is exact (i.e. the solution to the MILP is equal to the solution to the original optimization problem), the linear program is instead a *lower bound* on the solution to the original problem. We can do this in one of two ways:

- Relax the ReLU to to its convex outer bound (Figure **??**)
- Relax the ReLU to the interval constraint

The first is the tightest bound and consists of three linear constraints (triangle relaxation):

$$z \geq 0, \; z \geq \tilde{z}, \; -u\tilde{z} + (u - \ell)z \leq -u\ell \tag{4}$$

This is the basis of the linear programming bound. Another way to relax is looser but quick bound (rectangle relaxation)

$$\max(0, \ell) \leq z \leq \max(0, u) \tag{5}$$

which is often referred to as interval bounds. Different shapes can be taken here—an expansion of the traingle to a parallelogram gives rise to the duality-based bounds, while other slopes between 0 and 1 for the lower edge of the quadrangle result in other linear bound propagation methods.

3 References

Some notes taken from the classic Reluplex paper `https://arxiv.org/pdf/1702.01135.pdf`