CTRLVERIF. Analysis of control systems

Lecture 2. Abstraction-based verification of neural networks

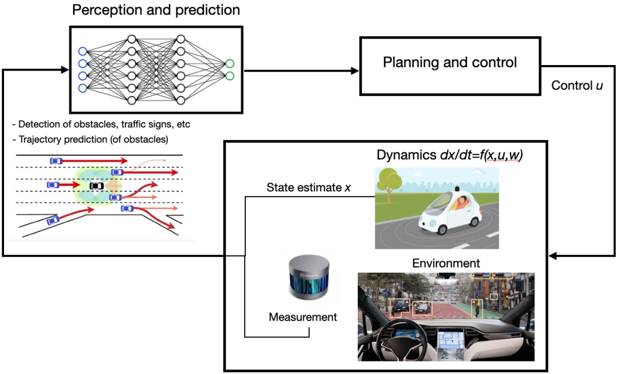
Eric Goubault and Sylvie Putot

MPRI

Outline

- Introduction to the safety verification of neural network
- A word on (complete) constrained-based approaches
- (Incomplete) abstraction-based forward reachability
- Going further: quantitative/probabilistic verification, backward reachability

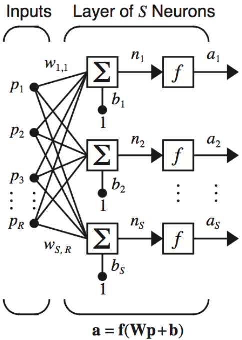
Neural networks in autonomous systems

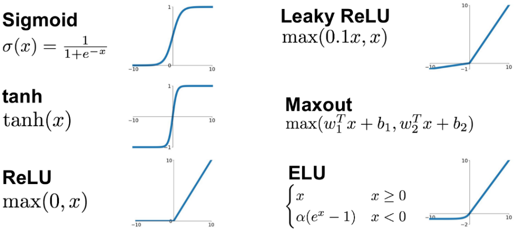


- Perception: object detection and classification (obstacle, lane marking, etc)
- Planning and control:
  - Neural network as function approximator: predict the response of a nonlinear plant over a time horizon (system/model identification, e.g. for dynamics either too complex to model or uncertain)
  - Neural network as controller (e.g. trained to approximate a traditional controller)
- End-to-end learning: a unique network for the system, from sensors to actuators

Feedforward neural networks (the simplest!)

- Succession of layers (inner ones are “hidden”) consisting of simple neurons
- Each layer = a linear transform followed by a non linear activation function





Universal approximation guarantee

Can approximate a continuous function on a compact set to arbitrary accuracy

Specifying and verifying neural networks

RELU activation function

- Very commonly used in applications, and the first target of verification approaches
- Piecewise linear activation function: easy encoding to linear arithmetic constraints
- Each neuron is conceptually a switch (2^N configurations): verification NP-complete

Apply traditional program verification ? But:

- Neural networks have specific structure and are often very large: specific and scalable abstractions
  - for feedforward networks, straight line code: no fixpoint computations
- Their internal workings are not perfectly understood:
  - they are learned from data instead of written by a human being
  - a specific weight or part of a network cannot be pointed out as the cause of a behavior
  - local/compositional reasoning almost impossible?
- Many applications lack specifications
  - if the network must recognize a stop sign, how do we mathematically specify?

Specifying and verifying neural networks

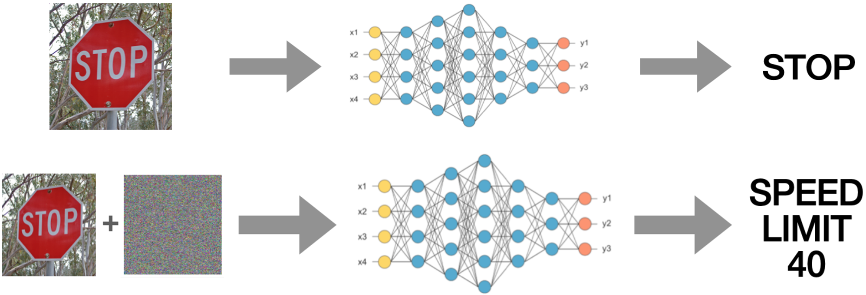
What can we do?

- Robustness to disturbances
- Safety properties when available: input-output relationships
- Closed-loop properties (reach-avoid properties, stability, invariance sets etc)
- But also other properties such as fairness (prediction being independent of sensitive input values) - see related MPRI course on abstract interpretation

Today: focus on robustness and safety of feedforward neural networks

Robustness to adversarial disturbances

Perception: objects (obstacles, traffic sign, etc.) detection should be robust to change in lighting, physical attacks, adversarial noise



Source: Eykholt et al, Robust Physical-World Attacks on Deep Learning Visual Classification, 2018 If the NN has n outputs NN_1 to NN_n, the property that every image is classified to $i \in [1, n]$ writes:

$$\forall j \in [1, n], NN_i(x) \geq NN_j(x)$$

Local robustness

Example of safety properties provided by input-output relationships

Example of ACAS Xu: collision avoidance systems for civil aircrafts (FAA)

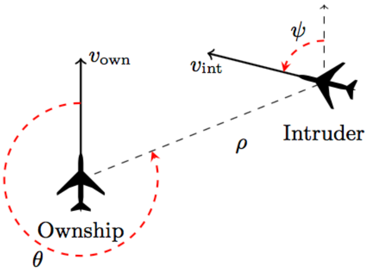
- New traffic alert and collision avoidance system : ACAS X (Airborne Collision Avoidance System X)
- Given information on relative position of an intruder aircraft with respect to the plane
- Produces aicraft advisory (clear-of-conflict, weak right, weak left, strong right, etc.)
- Unmanned version : ACAS Xu, large lookup table of about 2GB.
- DNN representation proposed as a replacement

Ref: *Deep Neural Network Compression for Aircraft Collision Avoidance Systems, Julian et al. 2018*

ACAS Xu

Inputs:

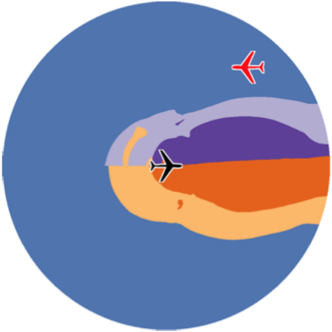
- ρ : Distance from ownship to intruder
- θ : Angle to intruder relative to ownship heading direction;
- ψ : Heading angle of intruder relative to ownship heading;
- v_{own} : Speed of ownship;
- v_{int} : Speed of intruder;
- τ : Time until loss of vertical separation;
- a_{prev} : Previous advisory.



ACAS Xu representation by DNNs

45 DNNs

- Produced by discretizing τ and a_{prev} ; each one has 5 inputs ($\rho, \theta, \psi, v_{own}$ and v_{int}) and 5 outputs (score for COC, weak right, weak left, strong right, strong left).
- Each DNN is fully connected with 6 hidden layers (300 RELU nodes each DNN).



Advisory as function of (ρ, θ)
from SyReNN: A Tool for Analyzing Deep Neural Networks, M. Sotoudeh and A. V. Thakur, 2021

Sample property to verify:
If the intruder is distant and is significantly slower than the ownship, network advises clear of conflict

$$\rho \geq 55947, v_{own} \geq 1145, v_{int} \leq 60 \implies \dots$$

Neural Network Verification

All these properties:

- Need to be proved for (possibly large) sets of network inputs
- Can be specified as preconditions/postconditions expressed in linear arithmetic

Two classes of approaches

- Complete constraint-based approaches
- Incomplete abstraction-based approaches: our focus in the context of control systems

Constraint-based verification approaches

Encode neural network and verification conditions as constraints

- For ReLU activations, input-output relations on neural networks can be encoded as constraints in Linear Real Arithmetic (LRA)

  ex. for ACAS Xu: $|v_{own} - v_{int}| \leq 10 \wedge \rho < 45 \wedge \theta \in [0, \pi/4] \wedge \ldots) \implies score(coc) \geq 0.8 \ldots$

Send these constraints to solvers

- SMT solvers (Z3, MathSAT, Yices)
- or Mixed Integer Linear Programming (MILP) solvers (Gurobi, Mosek, GLPK)

Complete verification

- If the property holds, then the method is able to prove it
- However, this comes at a cost (NP-complete)

Satisfiability Modulo Theory (SMT) Encoding

SAT/SMT

- SAT solving: satisfiability of a formula expressed in a logic of predicates
  ex. $f = (v_1 \vee v_2) \wedge (\neg v_1 \vee v_3) \wedge (\neg v_2 \vee \neg v_1)$
- SMT solving: satisfiability of a formula expressed in a logic of predicates+axioms defining a theory
  ex. $f = (x < 0 \vee x > 1) \wedge (x = y + 5) \wedge (y > 0)$ for the theory of real numbers
- SAT/SMT solving is NP-complete (exp-time), but there are efficient algorithms in practice (DPLL etc.)

Linear Real Arithmetic and the Simplex algorithm

- Signature $\{+, -, ., \leq, \geq\}$ and standard model for real numbers
- Linear formulas

The simplex algorithm is an efficient procedure for deciding whether a linear formula can be satisfied in real numbers, or not.

SMT encoding of neural network verification

Naive approach

- encode $z = RELU(y)$ using disjunctions and send to a SMT solver:

$$(y = \sum_i \omega_i x_i \wedge y \leq 0 \wedge z = 0) \vee (y = \sum_i \omega_i x_i \wedge y \geq 0 \wedge z = y)$$

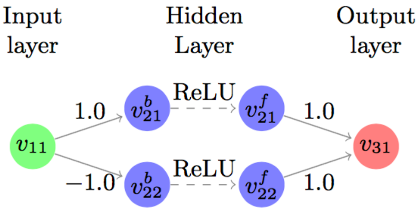
- fails to handle networks beyond a few dozens of neurons

Reluplex: Extension of linear real arithmetic for RELU

- Add in the signature $ReLU(x, y)$ with interpretation $ReLU(x, y)$ iff $y = max(0, x)$
- Extend simplex algorithm with Relu ($y = \sum_i \omega_i x_i \wedge ReLU(x, y)$) + branch and bound
- Key idea = try to delay case splitting on ReLUs by natively handling them

Reluplex : An Efficient SMT Solver for Verifying Deep Neural Networks, G. Katz et al., CAV 2017.

Reluplex on a simple example



Property we want to check

Is it possible to satisfy $v_{11} \in [0, 1]$ and $v_{31} \in [0.5, 1]$?

Encoding

- Encode each ReLU node using pair of variables v^b, v^f such that $ReLU(v^b, v^f)$
- Simplex: new basic variables to encode the linear transforms between nodes: $a_1 = -v_{11} + v_{21}^b$, $a_2 = v_{11} + v_{22}^b$, $a_3 = -v_{21}^f - v_{22}^f + v_{31}$ (with $a_1 = a_2 = a_3 = 0$)

Reluplex on a simple example (continued)

Is it possible to satisfy on the network $v_{11} \in [0,1]$ and $v_{31} \in [0.5,1]$?

$a_1 = -v_{11} + v_{21}^b, a_2 = v_{11} + v_{22}^b, a_3 = -v_{21}^f - v_{22}^f + v_{31}$ (with $a_1 = a_2 = a_3 = 0$)

As for simplex, iterative process to search for a feasible variable assignment:

- variables can temporarily violate their bounds or the ReLU semantics
- iteratively correct variables that are either out of bounds or pairs violating a ReLU

Initially: bounds defined by the problem (hidden var. unconstrained), assignment to 0:

variable	v_{11}	v_{21}^b	v_{21}^f	v_{22}^b	v_{22}^f	v_{31}	a_1	a_2	a_3
lower bound	0	$-\infty$	0	$-\infty$	0	0.5	0	0	0
assignment	0	0	0	0	0	0	0	0	0
upper bound	1	∞	∞	∞	∞	1	0	0	0

First fix out-of-bounds variables (v_{31}) then pivot (v_{21}^f), update (v_{21}^b), pivot (v_{11}), reaching a feasible solution:

variable	v_{11}	v_{21}^b	v_{21}^f	v_{22}^b	v_{22}^f	v_{31}	a_1	a_2	a_3
lower bound	0	$-\infty$	0	$-\infty$	0	0.5	0	0	0
assignment	0.5	0.5	0.5	-0.5	0	0.5	0	0	0
upper bound	1	∞	∞	∞	∞	1	0	0	0

MILP encoding

A Mixed Integer Linear Program (MILP) is of the form

$\min c^T x$		objective function
$Ax \leq b$		linear constraints
$x \geq 0$	(or $l \leq x \leq u$)	bounds
$x_i \in \boldsymbol{Z}, \forall i \in I$	some x_i are integers	used for ReLU encoding

Neural network verification encoding:

- objective function: post-condition
- linear constraints: pre-condition and affine layers
- bounds: input domain + first estimate by box propagation
- MILP encoding of $y = ReLU(x) = \max(0, x)$ (can be refined with variables bounds):

$$\text{large constant } M_x, \text{ binary variable } \delta_x \in \{0, 1\}$$

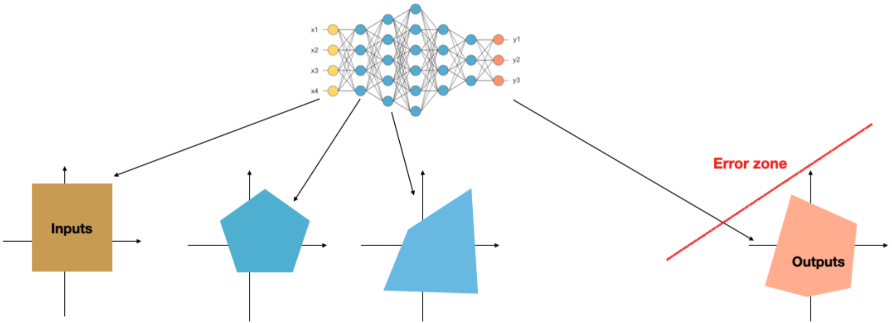
$$y \geq x\,, \quad y \geq 0$$

$$y \leq x + (1 - \delta_x) M_x\,, \quad y \leq \delta_x M_x$$

Reachability Analysis for Neural Network Verification

Abstraction based verification

- ReLU networks are piecewise linear, but we don't want to decompose all sub-regions
- Usually abstract layer by layer, and often neuron by neuron, avoiding disjunctions: scalable but possibly (very) conservative
- Relying on abstraction from program analysis, customized for neural networks



Propagating sets through NN: the Box abstraction

Box (or Hyperrectangle)

For a vector of variables $x \in \mathbb{R}^n$, a Box is a Cartesian product of n Intervals $[a_i, b_i] = \{x_i \in \mathbb{R}, x_i \geq a_i \wedge x_i \leq b_i\}$ for each $x_i,\ i = 1, \ldots, n$.

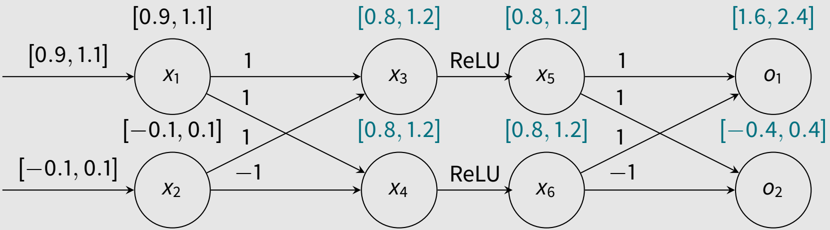
Abstract transformers on Intervals

For $a, b, c, d \in \mathbb{R}^n$ and $\lambda > 0$:

$$
\begin{aligned}
&[a, b] +^{\#} [c, d] = [a + c, b + d] \\
&-^{\#} [a, b] = [-b, -a] \\
&\lambda^{\#}[a, b] = [\lambda a, \lambda b] \\
&ReLU^{\#}[a, b] = [ReLU(a), ReLU(b)]
\end{aligned}
$$

The Box abstraction can be used to prove some properties

Example (Verifying Robustness)



Robustness problem:

- for a given input vector $x = (x_1, x_2)$ (think of the pixel values of an image), the result is classified by the network with class 1 if $o_1(x) \geq o_2(x)$, otherwise 2.
- do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.1 on all components ?
- YES: $\forall o_1 \in [1.6, 2.4], o_2 \in [-0.4, 0.4]$, we always have $o_1(x) \geq o_2(x)$.

But Boxes can also fail to prove some true properties

Do boxes succeed in verifying local robustness around $(x_1, x_2) = (1.0, 0.0)$ for a maximal perturbation of 0.3 on all components ?

Zonotopes, remember

Definition (Zonotope)

An n-dimensional zonotope $\mathcal{Z}$ with center $c \in R^n$ and a vector $G = \left[g_1 \dots g_p\right] \in \mathbb{R}^{n,p}$ of p generators $g_j = (g_{ij})_{i=1,\dots,n} \in \mathbb{R}^n$ for $j = 1, \dots, p$ is defined as $\mathcal{Z} = \langle c, G \rangle = \{c + G\varepsilon \mid \|\varepsilon\|_\infty \leq 1\}$.

In other words, for every dimension $1 \leq i \leq n$ we have the ith coordinate z_i of points $z \in \mathcal{Z}$ that belongs to the set:

$$z_i = \{c_i + \sum_{j=1}^{p} g_{ij}\varepsilon_j \mid \varepsilon \in [-1, 1]^p\}$$

Zonotopes are closed under affine transformations:

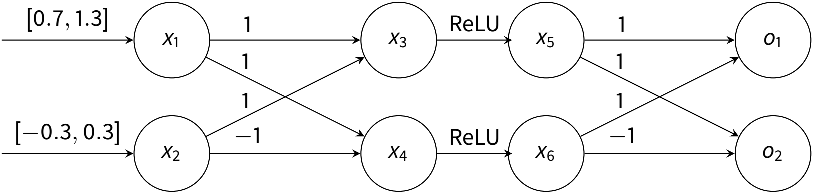
For $A \in \mathbb{R}^{m,n}$ and $b \in \mathbb{R}^m$ we define

$$A\mathcal{Z} + b = \langle Ac + b, AG \rangle$$

as the m-dimensional resulting zonotope.

Coming back to the example where Boxes fail to prove robustness

Exercise: do zonotopes succeed in verifying robustness ?



Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

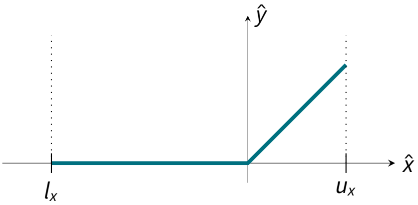
For

$$\hat{x} = x_0 + x_1\varepsilon_1 + \ldots + x_p\varepsilon_p, \ \varepsilon_1, \ldots \varepsilon_p \in [-1, 1]$$

we can bound the values reachable by x by

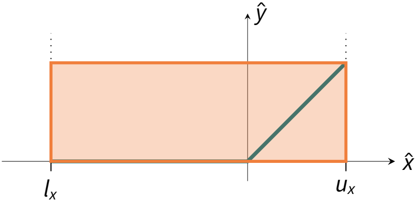
$$[l_x, u_x] = [x_0 - \sum_{i=1}^{p} |x_i|, x_0 + \sum_{i=1}^{p} |x_i|]$$

- if $l_x \geq 0$ then $\hat{y} = \hat{x}$
- if $u_x \leq 0$ then $\hat{y} = 0$
- otherwise ?



Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

First option: a Box

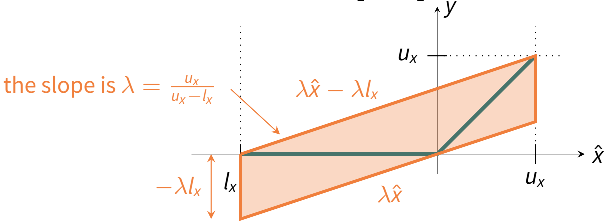


Zonotope transformer for RELU $\hat{y} = max(0, \hat{x})$

Second option: for fixed parametrization of the input $\hat{x}$, a zonotope for the output with minimal area in (x, y).

Fast and Effective Robustness Certification, Singh et al., NIPS 2018.

- parallel lines (otherwise not a zonotope), for fixed $\hat{x}$, 2 vertical faces
- parameterized by $\lambda = \frac{u_x}{u_x - l_x}$: lower line is $\lambda\hat{x}$, upper line is $\lambda\hat{x} - \lambda l_x$
- from $\lambda\hat{x} \leq \hat{y} \leq \lambda\hat{x} - \lambda l_x$, deduce $\hat{y} = \lambda\hat{x} - \frac{\lambda l_x}{2} - \frac{\lambda l_x}{2}\varepsilon_{new}$

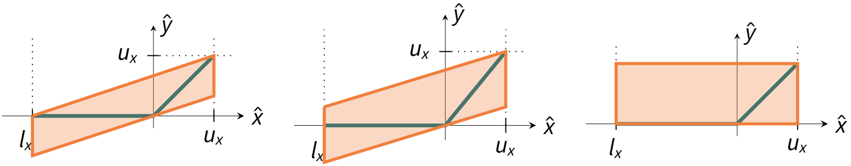


Example :

$\hat{x} = -0.5 + 1.5\varepsilon_1,$

Varying the slope λ

- There are many non comparable Zonotope transformers: not one zonotope is smaller in terms of included in the others
- Even the Box transformer is not strictly comparable
- The one of the previous slide is minimal in term of area in the input-output plane



$$\lambda = \frac{u_x}{u_x - l_x} = \frac{1}{3} \qquad \lambda = \frac{1}{4} \qquad \lambda = 0$$

References:

- AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, M. Vechev, IEEE S&P 2018
- Fast and Effective Robustness Certification, G. Singh, T. Gehr, M. Mirman, M. Püschel, M. Vechev, NIPS 2018.

Other Zonotopes transformers for RELU are possible

Can you imagine another possibly interesting zonotope transformer ?

What about other activation functions?

Exercice: define a zonotope transformer for the sigmoid function

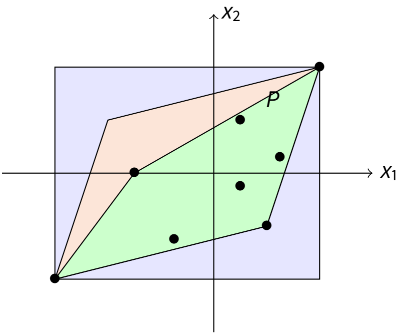
$$y = \sigma(x) = \frac{e^x}{1 + e^x}$$

Numerical abstract domains

We have seen:

- Intervals/Boxes/Hyperrectangles (synonymous)
- Zonotopes

Now let us see Convex Polyhedra.



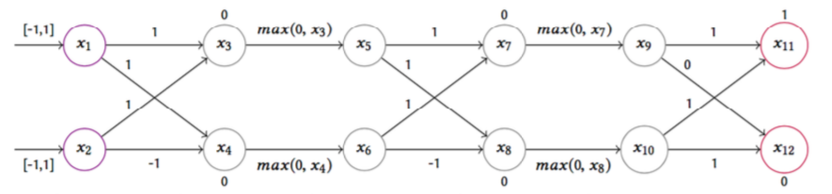
Convex Polyhedra abstractions

The problem is similar to what we have already seen:

Example

Proving specifications such as

- Two inputs: $x_1 \in [-1, 1]$ and $x_2 \in [-1, 1]$, two outputs x_{11} and x_{12}
- Specification: $\forall x_1, x_2 \in [-1, 1]$, we always have $x_{11} \geq x_{12}$ (classification problem)



The Convex Polyhedra abstraction ([Cousot& Halbwachs 1979])

Abstraction by Polyhedra P for Program Analysis usually rely on a double description:

- Constraint representation : an intersection of a finite number of closed half spaces of the form $a^T x \leq \beta$ and a finite number of subspaces of the form $d^T x = \xi$, i.e.

$$P = \{x \in \mathbb{R}^n | Ax \leq b \text{ and } Dx = e\}$$

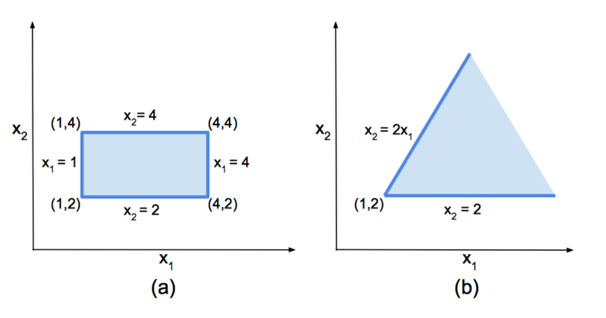
- Generator representation : a convex hull of a finite set of vertices v_i, a finite set of rays r_j and a finite set of lines z_k, i.e. $x \in P$ iff :

$$x = \sum_{i=1}^{u} \lambda_i v_i + \sum_{j=1}^{v} \mu_i r_i + \sum_{i=1}^{w} \nu_i z_i$$

where $\lambda_i, \mu_i \geq 0$ and $\sum_{i=1}^{u} \lambda_i = 1$.

Chernikova's algorithm is used to convert between the above representations (but this has worst case exponential complexity!)

Example of the double description



In equations

- Left : $C = \{-x_1 \leq -1, x_1 \leq 4, -x_2 \leq -2, x_2 \leq 4\}$ or $G = \{V = \{(1,2), (1,4), (4,2), (4,4)\}, R = \emptyset, Z = \emptyset\}$
- Right : $C = \{-x_2 \leq -2, x_2 \leq 2x_1\}$ or $G = \{V = \{(1,2)\}, R = \{(1,2), (1,0)\}, Z = \emptyset\}$.

Abstract operators

Order-theoretic operations

- Join : $P \cup Q$ is the convex hull of P and Q (easy with the vertex representation)
- Meet : $P \cap Q$ is obtained using the constraint representation, by concatenating the constraints of P and Q
- Inclusion : $P \subseteq Q$ is implemented using LP (linear programming). For each constraint $\sum a_i x_i \leq b$ in Q, compute $\mu = max \sum a_i x_i$ subject to constraints of P : if $\mu > b$ the inclusion does not hold

Arithmetic operations

- Linear assignments $x = L$: add a new variable x to P and the constraint $x - L = 0$ (then use Chernikova for getting the vertex set representation)
- Non linear assignments : generally by linearization

The DeepPoly convex relaxation

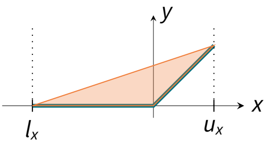
Ref. An Abstract Domain for Certifying Neural Networks, G. SIngh, T. Gehr, M. Puschel, M. Vechev, in POPL 2019

For each neuron (or variable) x_i:

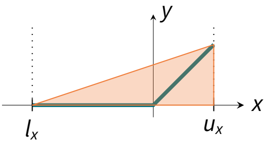
- upper and lower bounds: $x_i \leq u_i$ and $\boldsymbol{x}_i \geq l_i$
- two polyhedral constraints $x_i \leq \sum_j u_{ij} x_j + u_{i0}$ and $x_i \geq \sum_j l_{ij} x_j + l_{i0}$ where the x_j only refer to "previous" variables in the network.

- A restriction of Polyhedra to ensure scalability
- Affine transforms are exact (and easy)
- Custom convex relaxations for activation functions
- Generally more accurate but more costly than Zonotopes

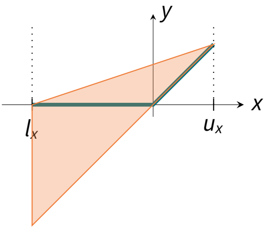
Abstract Transformers: ReLU activation



Optimal Convex transformer (triangle abstraction)



DeepPoly transformer 1



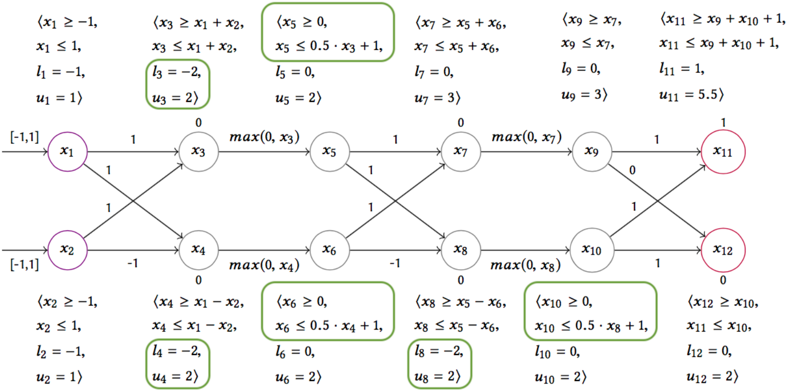
DeepPoly transformer 2

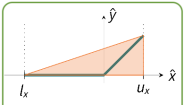
- Upper constraint

$$y \leq \lambda x + \mu,$$

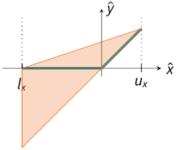
  with $\lambda = \frac{u_x}{(u_x - l_x)}$ and $\mu = \frac{-l_x u_x}{(u_x - l_x)}$.
- Optimal (triangle) transformer contains two lower polyhedral constraints for y, which is not allowed by the restricted domain
- Choice between RELU transformers 1 or 2 depends on area (heuristic): both are smaller area-wise than the Zonotope transformer

Analysis by DeepPoly on the example: ReLU transformer



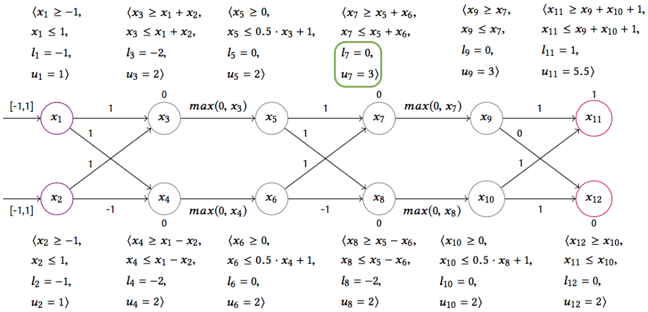


DeepPoly transformer 1



DeepPoly transformer 2

Analysis by DeepPoly on the example: affine transformers

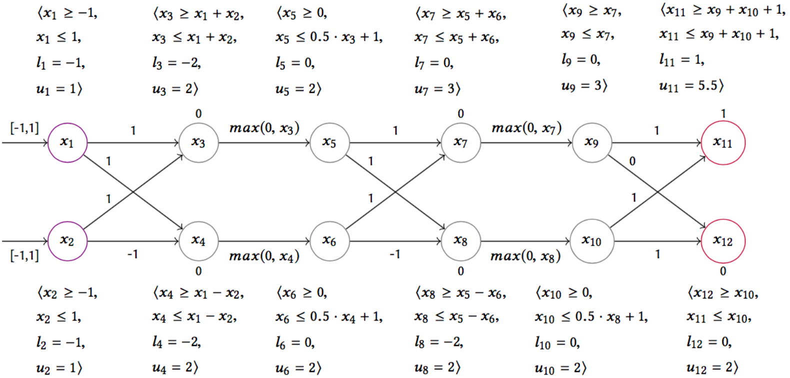


Precise bounds (useful for ReLU) by backsubstitution on the polyhedral constraints:

$$
\begin{array}{ccccc}
x_5 + x_6 & \leq & x_7 & \leq & x_5 + x_6 \\
0 & \leq & x_7 & \leq & 0.5x_3 + 0.5x_4 + 2 \\
0 & \leq & x_7 & \leq & 0.5(x_1 + x_2) + 0.5(x_1 - x_2) + 2 \\
0 & \leq & x_7 & \leq & x_1 + 2 \leq 3
\end{array}
$$

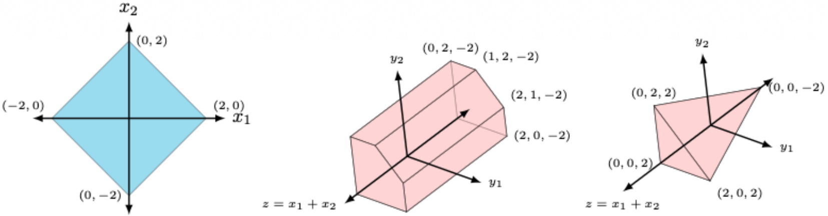
Checking the specification

Check whether $\forall i_1, i_2 \in [-1,1] \times [-1,1], x_{11} \geq x_{12}$ (robustness of classification) ?



Abstracting each neuron separately is not optimal

Example: $y_1 =$ReLU(x_1) and $y_2 =$ReLU(x_2) starting from x_1, x_2 with $x_2 - x_1 \leq 2$, $x_1 - x_2 \leq 2, x_1 + x_2 \leq 2, -x_1 - x_2 \leq 2$



(a) Input shape (b) 1-ReLU (c) 2-ReLU

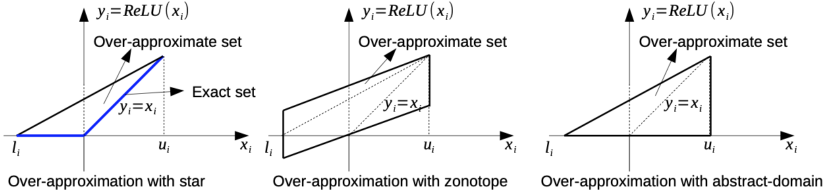
- 1-ReLU computes independent. triangles $y_1 \geq 0, y_1 \geq x_1, y_1 \leq 0.5x_1 + 1$ and (x_2, y_2)
- k-ReLU: abstract jointly the output of multiple Relus instead of separately = exploit relations between x_1 and x_2 to deduce relations between y_1 and y_2
- Instantiated for polyhedra but applies to other abstract domains: how would you handle the case of zonotopes ?

- Beyond the single neuron convex barrier for neural network certification, 2019, Singh et al.

- PRIMA: General and Precise Neural Network Certification via Scalable Convex Hull Approx, 2022, Müller et al.

Many refinements of zonotopes

- Star sets $\mathcal{Z} = \langle c, G \rangle = \{c + G\varepsilon \mid \|\varepsilon\|_\infty \leq 1 \wedge C\varepsilon \leq d\}$: extension of zonotopes with constraints, that can be as expressive as Polyhedra:
  - a convenient representation of polyhedra (zonotope-style transfer functions for affine layers)
  - both exact and over-approximated propagation algorithms



Star-based reachability analysis of deep neural networks, Tran et al., 2019.

- RefineZono: combined zonotope abstraction and MILP encoding:

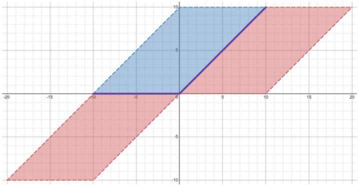
  Boosting Robustness Certification of Neural Networks, 2019, Singh, Gehr, Püschel, Vechev

Many refinements of zonotopes (end)

Hybrid zonotopes: constrained zonotopes + encode disjunction with discrete noise symbols in $\{-1, 1\}$

$$\mathcal{Z}_h = \left\{ \begin{bmatrix} G^c & G^b \end{bmatrix} \begin{bmatrix} \xi^c \\ \xi^b \end{bmatrix} + c \;\middle|\; \begin{matrix} \begin{bmatrix} \xi^c \\ \xi^b \end{bmatrix} \in \mathcal{B}_\infty^{n_g} \times \{-1,1\}^{n_b}, \\ \begin{bmatrix} A^c & A^b \end{bmatrix} \begin{bmatrix} \xi^c \\ \xi^b \end{bmatrix} = b \end{matrix} \right\}$$

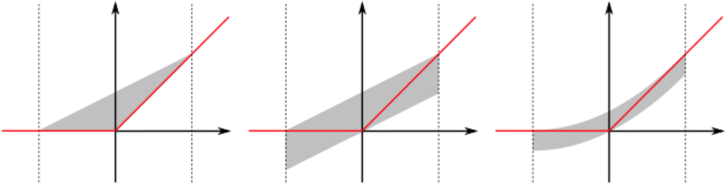
- the union of 2^{n_b} constrained zonotopes
- a hybrid zonotope with no binary generator is a constrained zonotope
- can represent exactly the ReLU by union of 2 zonotopes and 2 constraints



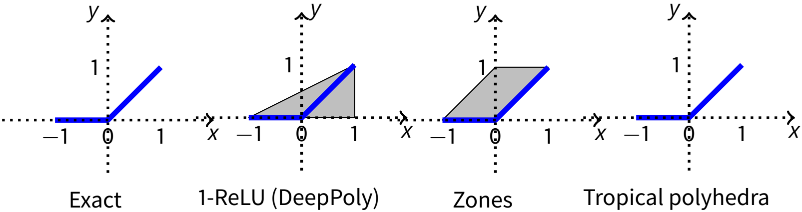
Hybrid Zonotopes Exactly Represent ReLU Neural Networks, Ortiz et al., 2023

Other abstractions: non-convex abstractions

- Polynomial Zonotopes *Open- and Closed-Loop Neural Network Verification using Polynomial Zonotopes, N. Kochdumper, C. Schilling, M. Althoff, and S. Bak, 2022*



- Max-plus or tropical polyhedra: ReLU $x \to max(x, 0) = x \oplus 0$ is tropically linear!



Static analysis of ReLU neural networks with tropical polyhedra, E. Goubault, S. Palumby, S. Putot, L. Rustenholz and S. Sankaranarayanan, SAS 2021

In summary: abstraction-based approaches

Incomplete but scalable methods:

- Compute output bounds by propagating the input domains through the network
- Abstract the range of outputs of neurons layer by layer (often neuron by neuron):
  - advantage: scales to large networks
  - drawback: is conservative, precision loss at each layer accumulate

Abstraction-based and constraint-based approaches can be combined to either scale complete methods or make incomplete methods more precise

- trade-off to find: how to choose and maintain some set disjunctions

Quantitative Neural Network Verification

Motivation

- Provide additional information on property satisfaction compared to SAT/UNKNOWN
- Often need quantitative, probabilistic guarantees on safety, security, reliability, performance, resource usage, etc, for instance
  - transportation: probability of a failure in a time interval should be less than 0.00001
  - neural network robustness: requiring no adversarial examples may be too strict, want high probability that local perturbations result in same classification result
- Exploit knowledge of probabilistic information on inputs
  - can be probabilistic but imprecisely known, e.g.:
    - Gaussian variable $\mathcal{N}(\mu, \sigma^2)$ with uncertain mean $\mu \in [\underline{\mu}, \overline{\mu}]$ and variance $\sigma^2 \in [\underline{\sigma^2}, \overline{\sigma^2}]$
    - Uniform variable $\mathcal{U}(a, b)$ with uncertain range (a and b uncertain)
  - example: noise due to sensor $V + \varepsilon$ with $V \in [a, b]$, ε a random variable

Problem Statement: propagating imprecise probabilities

Problem (Probability bounds analysis)

Given a ReLU network f and a constrained probabilistic input set

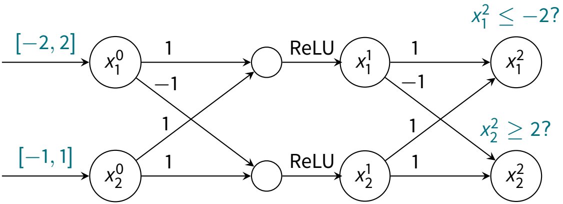
$$\mathcal{X} = \{X \in \mathbb{R}^{h_0} \mid CX \leq d \wedge \underline{F}(x) \leq \boldsymbol{P}(X \leq x) \leq \overline{F}(x), \forall x\}$$

where $\underline{F}$ and $\overline{F}$ are two cumulative distribution functions, compute a constrained probabilistic output set $\mathcal{Y}$ guaranteed to contain $\{f(X), X \in \mathcal{X}\}$.
For $X \in \mathbb{R}^n$, we note $\boldsymbol{P}(X \leq x) := \boldsymbol{P}(X_1 \leq x_1 \wedge X_2 \leq x_2 \ldots \wedge X_n \leq x_n)$

Problem (Quantitative property verification)

Given a ReLU network f, a constrained probabilistic input set $\mathcal{X}$ and a linear safety property $Hy \leq w$, bound the probability of the network output vector y satisfying this property.

Toy illustrating example: 2-layers ReLU network



Property:

- Qualitative: if $x^0 = \begin{bmatrix} x_1^0 & x_2^0 \end{bmatrix}^\top \in [-2, 2] \times [-1, 1]$, does output satisfy $x_1^2 \leq -2 \wedge x_2^2 \geq 2$?
- Quantitative:
  - $\boldsymbol{P}\left(x_1^2 \leq -2 \wedge x_2^2 \geq 2 \;\middle|\; x_1^0 \in \mathcal{U}(-2, 2) \wedge x_2^0 \in \mathcal{U}(-1, 1)\right)$?
  - $\boldsymbol{P}\left(x_1^2 \leq -2 \wedge x_2^2 \geq 2 \;\middle|\; x_1^0 \in \mathcal{N}(0, [0.5, 0.66]) \wedge x_2^0 \in \mathcal{N}([0, 1], 0.33)\right)$?

Representation of imprecise probabilities: P-box

Definition (P-box for a real-valued random variable X)

Given two (lower and upper) CDF (Cumulative Distribution Functions) $\underline{F}$ and $\overline{F}$ from $\mathbb{R}$ to $\mathbb{R}^+$ s.t. $\forall x \in \mathbb{R}, \underline{F}(x) \leq \overline{F}(x)$, the p-box $[\underline{F}, \overline{F}]$ represents the set of probability distributions for X s.t.

$$\forall x \in \mathbb{R}, \underline{F}(x) \leq \boldsymbol{P}(X \leq x) \leq \overline{F}(x).$$

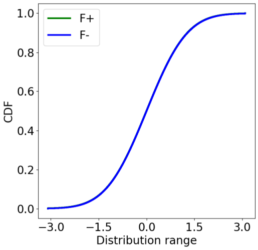
Ref:

- *Constructing probability boxes and Dempster–Shafer structures. Ferson et al., Tech. Rep. SAND2002-4015, 2003*
- *Probabilistic Arithmetic I: Numerical Methods for Calculating Convolutions and Dependency Bounds, Williamson and Downs, Journal of Approximate Reasoning, 1990*

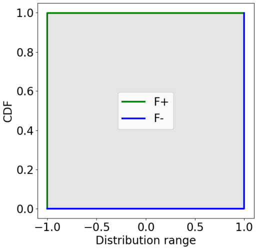
P-box examples (Julia library ProbabilityBoundsAnalysis.jl)l

Sets of probability distributions on X (CDF form) such that

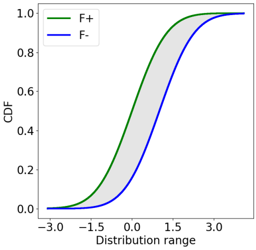
$$\forall x, F^{-}(x) \leq \boldsymbol{P}(X \leq x) \leq F^{+}(x) :$$



normal(0,1)



makepbox(interval(-1,1))



normal(interval(0,1),1)

Generalize probabilistic and non deterministic (interval) information

Dempster-Shafer structures (DSI)

Dempster-Shafer structure: a discrete version of a P-box

A finite set of focal elements with a probability mass:

$$d = \{\langle t_1, w(t_1)\rangle, \langle t_2, w(t_2)\rangle, \ldots, \langle t_N, w(t_N)\rangle\},$$

with $\boldsymbol{t}_i \in T$ and $w(t_i) \in [0, 1]$ its probability mass with $\sum_{i=1}^{N} w(t_i) = 1$.

- Focal elements ($\in T$ - here T is a set of subsets of $\mathbb{R}$):
  - sets of non-deterministic events/values
  - they usually overlap
- Weights associated to focal elements ($w : T \to \mathbb{R}^+$)
  - probabilistic information on the belonging to the focal elements

- A DSI defines a pbox which bounds are the Belief function *Bel* and Plausibility function *Pl* from $\wp(E)$ to $\mathbb{R}$:

$$Bel(S) = \sum_{t \in T, t \subseteq S} w(t) \leq P(S) \leq \sum_{t \in T, t \cap S \neq \emptyset} w(t) = Pl(S)$$

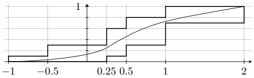
Dempster-Shafer Interval structures (DSI)

- Focal elements $t \in T$ (sets of values, here Intervals) with probability $w : T \to \mathbb{R}^+$

$t \in T$	[-1,0.25]	[-0.5,0.5]	[0.25,1]	[0.5,1]	[0.5,2]	[1,2]
$w(t)$	0.1	0.2	0.3	0.1	0.1	0.2

- Represents the set of probability distributions P on X such that:

$$\forall x \in [-1, -0.5],\ P(X \le x) \le 0.1\,,$$
$$\forall x \in [-0.5, 0.25],\ P(X \le x) \le 0.1 + 0.2\,,$$
$$\forall x \in [0.25, 0.5],\ 0.1 \le P(X \le x) \le 0.1 + 0.2 + 0.3\,,$$



etc.

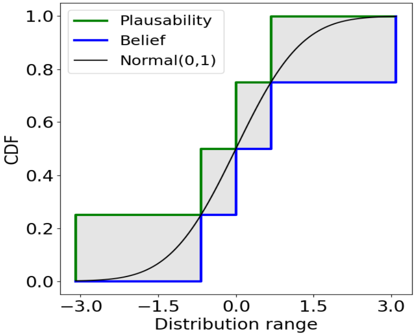
- They define Belief function *Bel* and Plausibility function *Pl* from $\wp(E)$ to $\mathbb{R}$:

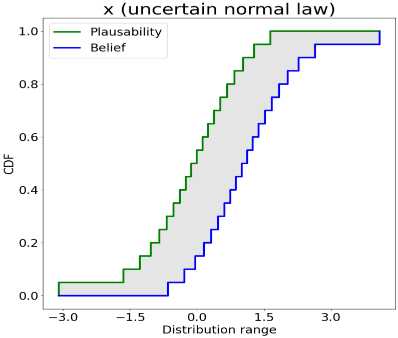
$$Bel(S) = \sum_{t \in T, t \subseteq S} w(t) \le P(S) \le \sum_{t \in T, t \cap S \neq \emptyset} w(t) = Pl(S)$$

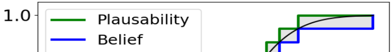
From P-boxes to Dempster-Shafer Interval structures

Given a P-box $(\underline{F}, \overline{F})$

- Take lower and upper approximation by stair functions
- Deduce focal elements (intervals) and weights





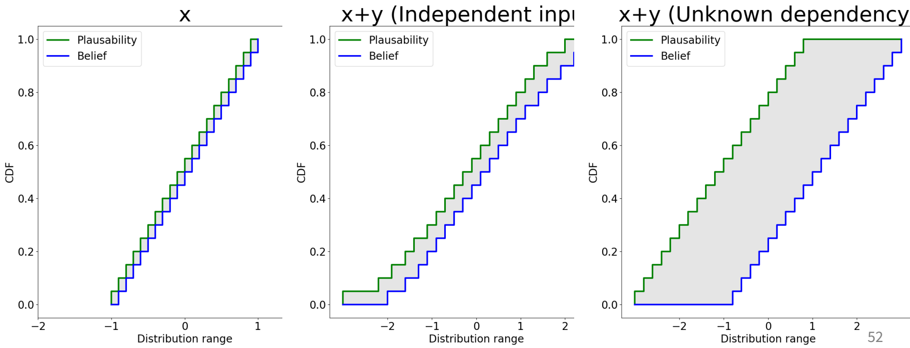


Arithmetic on DSI structures

DSI structures can be propagated through arithmetic operations:

- 2 simple cases: independent inputs / unknown dependency
- relying on interval arithmetic / Frechet inequalities
- conservative approximations

Can be generalized to multivariate dependency by adding external dependence information through copulas: compute multivariate law from marginals and copulas.



Arithmetic on DS structures: $z = x\square y$ ($\square$=+,-,$\times$,/ etc.)

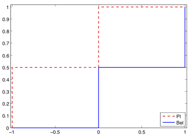
Independent variables x, y

- x (resp. y) given by focal elements T^x (resp. T^y) and weights w^x (resp. w^y)
- $T^z = \{t^x \square t^y \mid t^x \in T^X, t^y \in T^y\}$ and $w^z(t^x \square t^y) = w^x(t^x) w^y(t^y)$ (and renormalize)

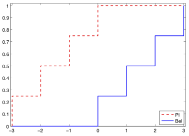
Example

- $T^x = \{[-1,0],[0,1]\}$, $w^x([-1,0]) = w^x([0,1]) = \frac{1}{2}$ (approximation of uniform distribution on [-1,1])
- $T^y = \{[-2,0],[0,2]\}$, $w^y([-2,0]) = w^y([0,2]) = \frac{1}{2}$

$x; y$	[-2,0], $\frac{1}{2}$	[0,2], $\frac{1}{2}$
[-1,0], $\frac{1}{2}$	[-3,0], $\frac{1}{4}$	[-1,2], $\frac{1}{4}$
[0,1], $\frac{1}{2}$	[-2,1], $\frac{1}{4}$	[0,3], $\frac{1}{4}$



CDF of x



CDF of $x + y$

Arithmetic on DSS for unknown dependencies (here $\square = +$)

- DS for x (similarly for y) given on $T^x = \{[a_i^x, b_i^x] \mid i = 1, \ldots, n\}$ by $w^x([a_i^x, b_i^x]) = w_i^x$
- Compute P-boxes for $z = x + y$ by LP using Frechet inequalities

Compute the stair functions given by values at $a_k^x + a_l^y$, $b_k^x + b_l^y$:

$$\overline{F}_z(a_k^x + a_l^y) = \min\left(\inf_{a_i^x + a_j^y = a_k^x + a_l^y} \sum_{i' \leq i} w_{i'}^x + \sum_{j' \leq j} w_{j'}^y, 1\right)$$

$$\underline{F}_z(b_k^x + b_l^y) = \max\left(\sup_{b_i^x + b_j^y = b_k^x + b_l^y} \sum_{i' \leq i} w_{i'}^x + \sum_{j' \leq j} w_{j'}^y - 1, 0\right)$$

ReLU

ReLU of a DSI

Given X represented by the DSI $\{\langle \boldsymbol{x_i}, w_i \rangle \ , \ i \in [1, n]\}$, then the CDF of $Y = \sigma(X) = \max(0, X)$ is included in the DSI

ReLU neural network analysis by DSI

Input: d^0 a h_0-dimensional vector of DSI

1: **for** $k = 0$ to $L-1$ **do**

2: **for** $l = 1$ to h_{k+1} **do**

3: $d_l^{k+1} \leftarrow \sigma(\sum_{j=1}^{h_k} a_{lj}^k d_j^k + b_l^k)$ ▷ *Affine transform and ReLU - Dependency graph useful for choosing the right DSI operations (indep. or unknown dep.) in affine transforms*

4: ***end for***

5: ***end for***

6: ***return*** $(d^L, cdf(Hd^L, w))$ ▷ *Vector of DSI for the output layer and probability bounds for property $Hz \leq w$*

Copula propagation can be used to refine the arithmetic in the above analysis (avoid unknown dependency operations that are used systematically starting at the 2nd layer).

Illustration on the toy example

Input $x^0 = \begin{bmatrix} x_1^0 & x_2^0 \end{bmatrix}^\top \in [-2, 2] \times [-1, 1]$ with Uniform law on inputs

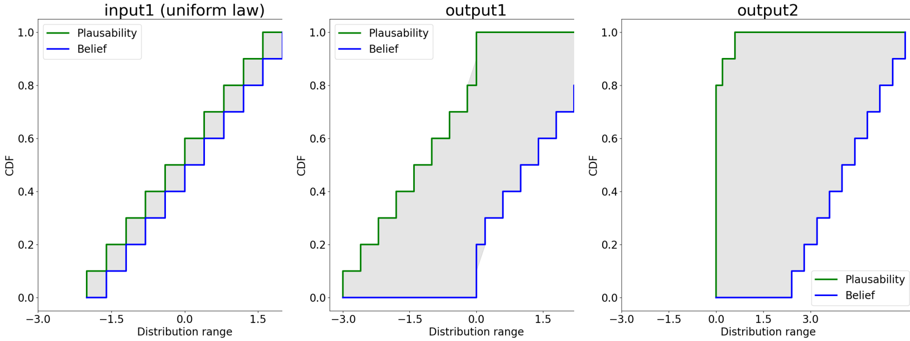
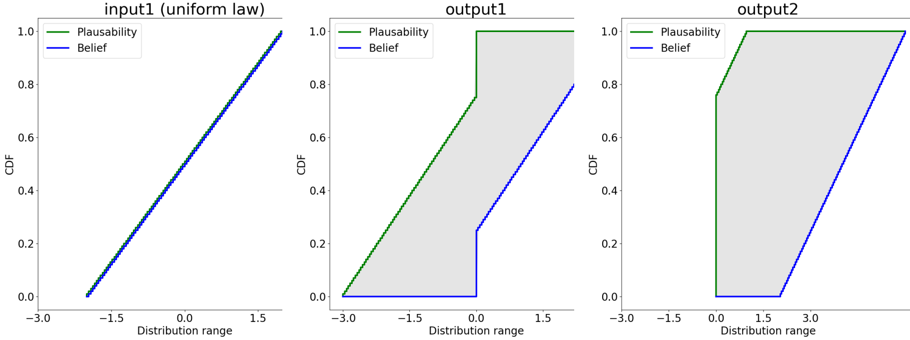


Illustration on the toy example

Input $x^0 = \begin{bmatrix} x_1^0 & x_2^0 \end{bmatrix}^\top \in [-2, 2] \times [-1, 1]$ with Uniform law on inputs



Finer discretization refines the approximation but the ranges are unchanged

Illustration on the toy example

Input $x^0 = \begin{bmatrix} x_1^0 & x_2^0 \end{bmatrix}^\top \in [-2, 2] \times [-1, 1]$ with Normal law on inputs

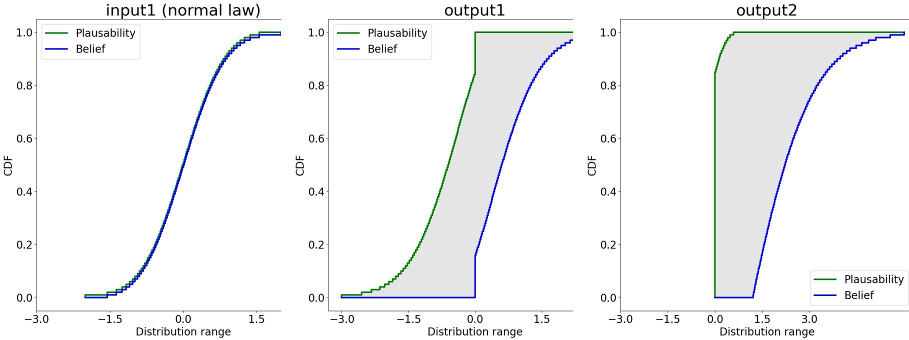
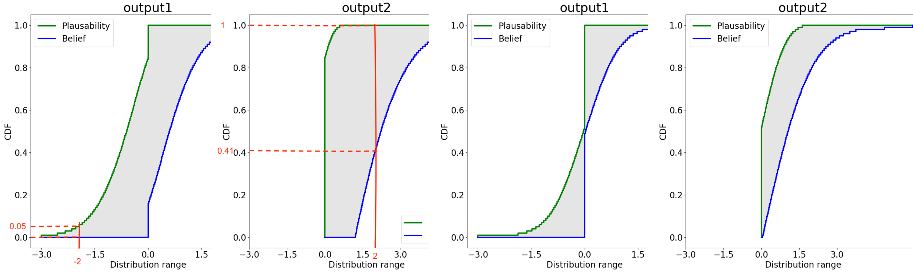


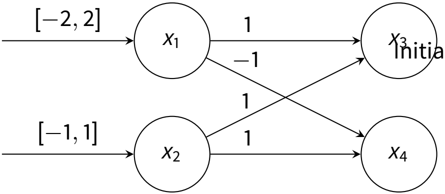
Illustration on the toy example

Unknown dependency on inputs vs independent inputs



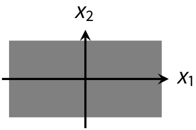
$\boldsymbol{P}(z_1 \leq -2) \in [0, 0.05]$ $\boldsymbol{P}(z_2 \geq 2) \in [0, 0.59]$ $\boldsymbol{P}(z_1 \leq -2) \in [0, 0.01]$ $\boldsymbol{P}(z_2 \geq 2) \in [0, 0.2]$

Wrapping effect: example of the first affine layer



Initial domain:

$$-2 \leq x_1 \leq 2$$
$$-1 \leq x_2 \leq 1$$

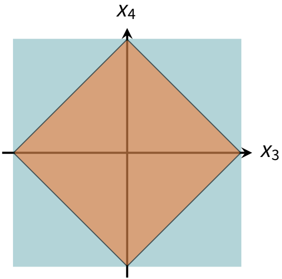


Exact domain:

$$x_3 = x_1 - x_2$$
$$x_4 = x_1 + x_2$$
$$x_1, x_2 \in [-1, 1]$$

Using Intervals/Boxes:

$$-3 \leq x_3 \leq 3$$
$$-3 \leq x_4 \leq 3$$
$$x_1, x_2 \in [-1, 1]$$



The optimal affine transformers for boxes are not exact. Zonotope transformers are !

Two solutions for zonotopic probabilistic NN analysis

Main idea: encode as much deterministic dependencies as possible by affine forms, and avoid/delay Dempster-Shafer arithmetic whenever possible

Probabilistic zonotopes (or probabilistic affine forms)

- Zonotopic network analysis starting from the support of input distribution
- Probabilistic interpretation: noise symbols are DSI instead of intervals

Dempster-Shafer Zonotopic structures (DSZ)

- Dempster-Shafer structures with zonotopic focal elements
  - initially boxes obtained by Cartesian product of interval focal elements from each input
  - propagation of each focal element in network by zonotopic analysis
- A refinement of probabilistic zonotopes, which fully exploits the DSI input
- As presented, restricted to independent inputs, but can be extended to general dependence using copulas
- Allows to obtain tight probability bounds on properties of the ACAS Xu benchmark

Ref: A Zonotopic Dempster-Shafer Approach to the Quantitative Verification of Neural Networks, E. Goubault and S. Putot, 2024

The inverse problem or backward reachability

Given a neural network N over set $\mathcal{X}$ of inputs and a property / set $\mathcal{Y}$ of outputs, compute the pre-image: all inputs $x \in \mathcal{X}$ such that $N(x) \in \mathcal{Y}$

Applications:

- Specification mining or rule extraction: some form of explanation of the function encoded
- Proving that some given specification holds: does there exist inputs leading to the set of outputs

Computing the pre-image of a ReLU network:

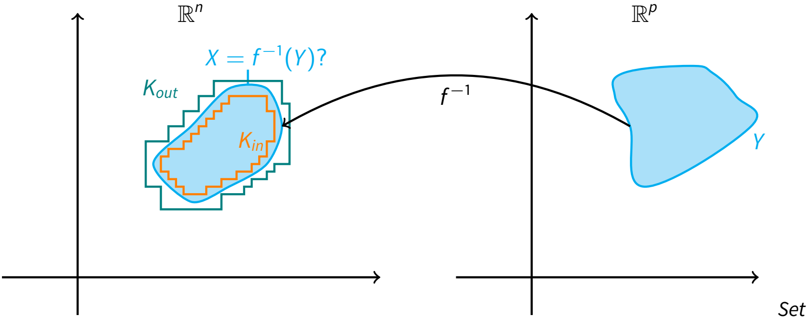
- Pre-image of a polyhedron by a ReLU network = a union of polyhedral sets
- Practically: in general inner-approximation of the pre-image?

Ref: The inverse problem for neural networks, M. Forets, C. Schilling, 2023

Set inversion problem

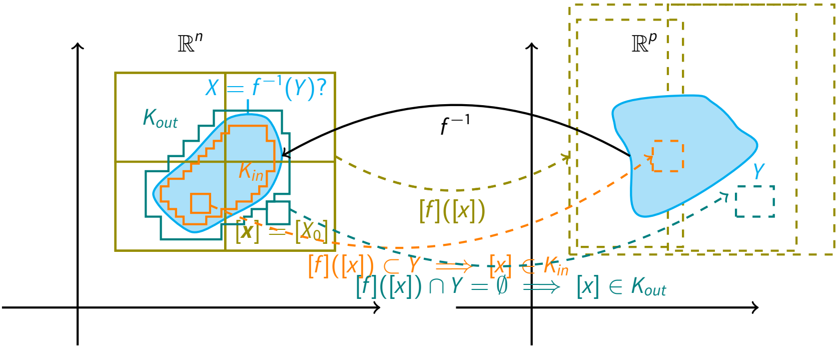
Let f be a continuous function from $\mathbb{R}^n$ to $\mathbb{R}^p$ and Y a compact subset of $\mathbb{R}^p$.

- characterize set $X = f^{-1}(Y)$.
- SIVIA (Set Inverter Via Interval Analysis): enclose X between subpavings K_{in} and K_{out}, such that $K_{in} \subset X \subset K_{out}$



Inversion via Interval Analysis for Nonlinear Bounded-error Estimation, L Jaulin and E. Walter, Automatica 1993

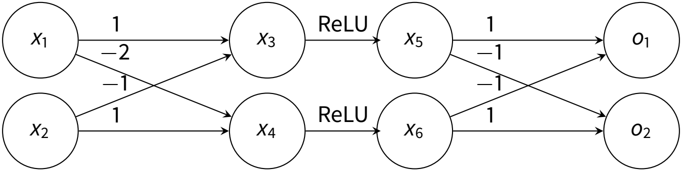
Set inversion via interval analysis (SIVIA)



Algorithm to compute K_{in} and K_{out}, given $Y \in \mathbb{R}^p$, f and initial region $[x] = [X_0] \subset \mathbb{R}^n$

- Feasible box: $[f]([x]) \subset Y \implies [x] \in K_{in}$
- Unfeasible box: $[f]([x]) \cap Y = \emptyset \implies [x] \in K_{out}$
- otherwise the box is indeterminate: bisect (different possible strategies) and try again on the bisected boxes until minimal box size reached

Example: visualing classes on input domain of a neural network



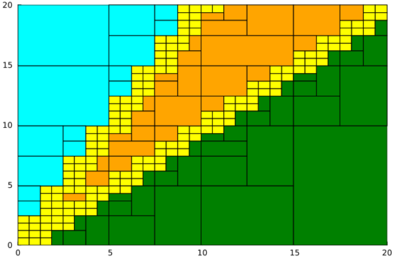
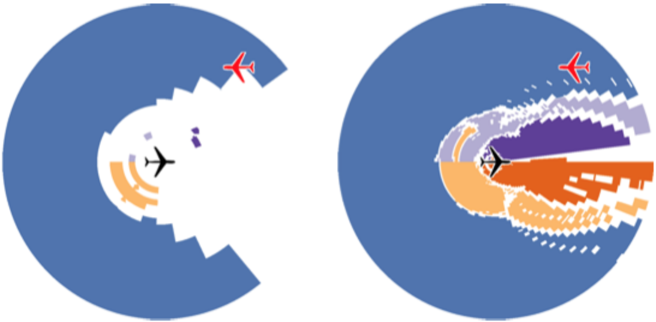


Figure 1: Interval-based paving of the (x_1, x_2) space: $o_1 > o_2$, $o_1 < o_2$, $o_1 = o_2$.

Will be refined with topological information in 2nd part of the course.

ACAS Xu advisory on input domain?

However, costly and should be used thoughtfully: below paving the input domain of the ACAS Xu (while a symbolic representation of a 2D subset of inputs yields precise results):



(b) Decision boundaries computed using DeepPoly$[k = 25^2]$

(c) Decision boundaries computed using DeepPoly$[k = 100^2]$

Ref. SyReNN: A Tool for Analyzing Deep Neural Networks, M. Sotoudeh and A. V. Thakur, 2021

Bibliography

The references in the slides

A survey and Julia library implementing many of these methods

- Algorithms for Verifying Deep Neural Networks, Liu et al. 2021: not very recent, but related to the below libraries
- NeuralVerification.jl and its documentation
- ModelVerification.jl: meant as a new version of the above (I have not tried it)

Next

For next week: paper reading

- *The inverse problem for neural networks, M. Forets, C. Schilling, 2023*
- Who is presenting ?

Next time: reachability verification of closed-loop systems