Verifying Neural Networks ReluPlex

Verifying cyberphysical systems

Sayan Mitra

mitras@illinois.edu

Reference : Introduction to Neural Network Verification by Aws Albarghouthi

Outline

Modeling neural networks

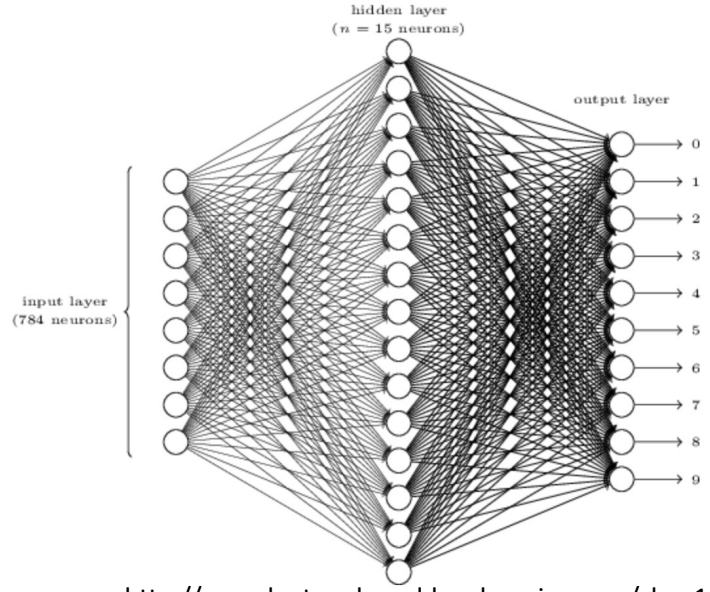
Requirements of neural networks

Reluplex algorithm

References

- Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks by Guy Katz, Clark Barrett, David Dill, Kyle Julian, Mykel Kochenderfer, CAV 2017
- Book: Introduction to Neural Network Verification by Aws Albarghouthi, 2021
- Further exploration: Neural Network Verification competition: https://sites.google.com/view/vnn20/
- **Neural Networks and Deep Learning**

Neural networks as graphs

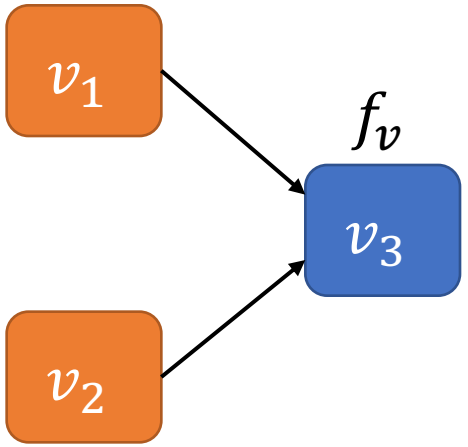


http://neuralnetworksanddeeplearning.com/chap1.html

- Consider a *network* $\mathrm{G} = (V, E)$
- Let $V^{in}, V^{o} \subseteq V$ be nonempty sets of input, output vertices
- Each $v \in V \setminus V^{in}$ is associated with a function $f_v \colon \mathbb{R}^{ind(v)} \to \mathbb{R}$ where $ind(v)$ is the indegree of v
- Typically these functions are differentiable (almost everywhere)

Assume w.l.o.g

- All nodes are reachable from some V^{in}
- Every node reach some V^{o}
- There is a total ordering on V and another on E

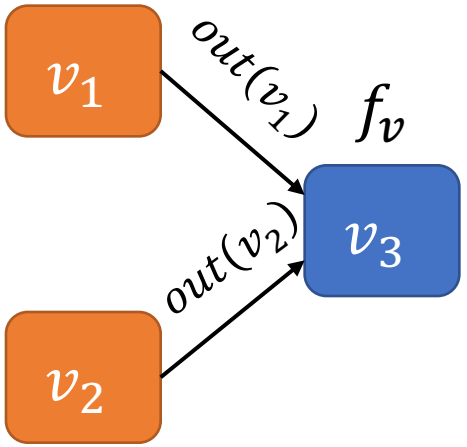


Computation performed by a Network

- $\left|V^{in}\right| = n, |V^{o}| = m$
- The network $G = (V, E)$ defines a function $\mathbb{R}^n \rightarrow \mathbb{R}^m$ defined as follows
- For any non input node $v \in V$ the output from v is recursively defined as follows: let $(v_1, v), (v_2, v), \ldots, \left(v_{ind(v)}, v\right)$ be the ordered set of all input edges to v

$$out(v) \coloneqq f_v\left(out(v_1), \ldots, out(v_{ind(v)})\right)$$

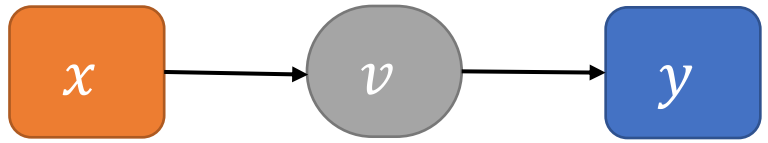
  - Base case: For input vector $\boldsymbol{x} \in \mathbb{R}^n$, for any input node $v \in V^{in}$, output $out(v) \coloneqq \boldsymbol{x}_i$



Example functions

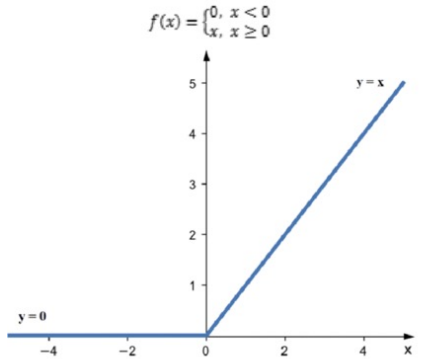
$$out(y) = f_y\left(f_v(out(x))\right)$$
$$= f_y(f_v(x))$$
$$= f_y(2x + 1)$$
$$= \max(0, 2x + 1)$$

$f_v \coloneqq 2in + 1$

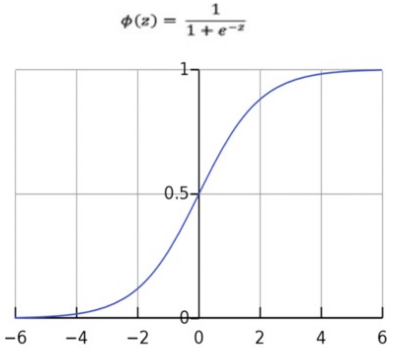


$f_y \coloneqq \max(0, \text{in})$

Rectified Linear Unit (RELU)



RELU

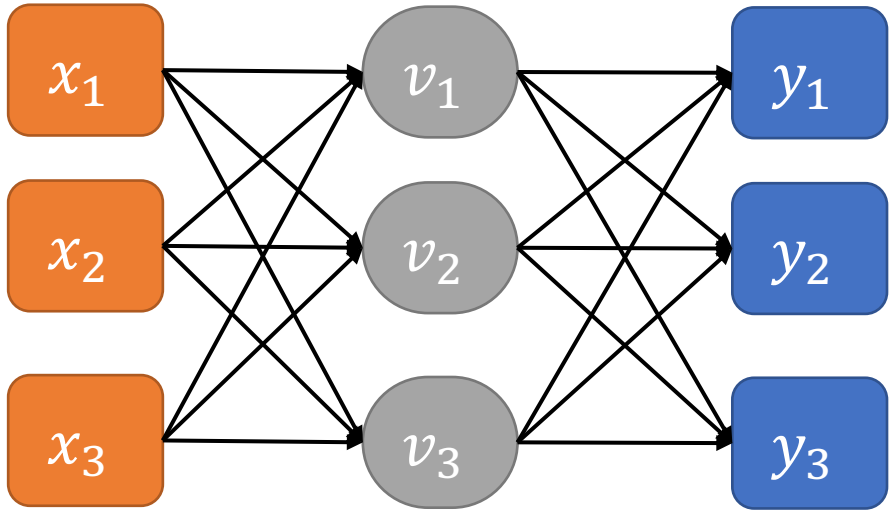


Sigmoid

$$f_y \coloneqq \sigma(\text{in}) = \frac{1}{1 + \exp(-in)}$$

Example functions

$$out(y_1) = f_y\left(\sum_{i=1}^{3} out(v_i)w_i\right)$$
$$= f_y\left(\sum_{i=1}^{3}\left(\sum_{j=1}^{3} x_j u_j\right)w_i\right)$$



Softmax

$$f_{y_i} := \frac{exp(v_i)}{\sum_{i=1}^{3} \exp(v_i)}$$

Multi-layered Perceptron

Hidden layer

Typical requirements studied for neural networks

For any input x (image, text, program, controller) $f(x)$ meets <some condition>

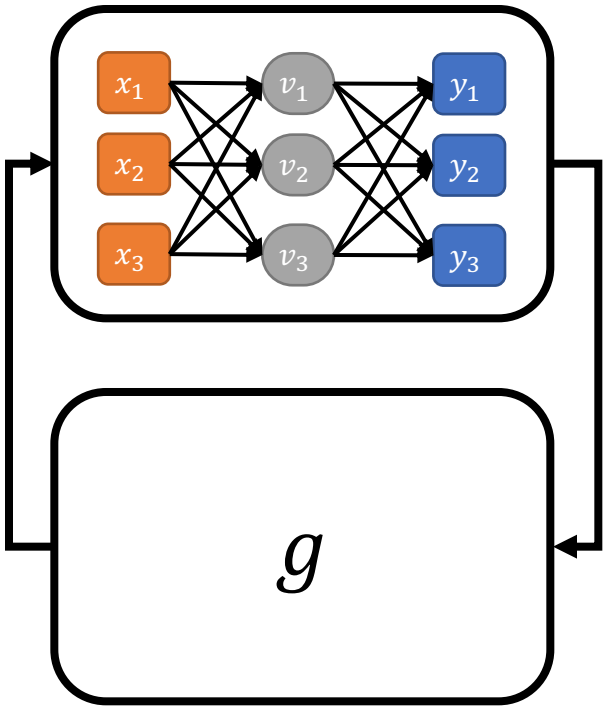
To check the validity of this statement, we can check the satisfiability of the negation, i.e., does there exist some input x such that $f(x)$ does not meet <some condition>

For any inputs x, y (image, text, program, controller) $f(x)$ and $f(y)$ meets <some condition>

Pros. Same form as pre-postconditions of automata transitions

Cons. with this framing

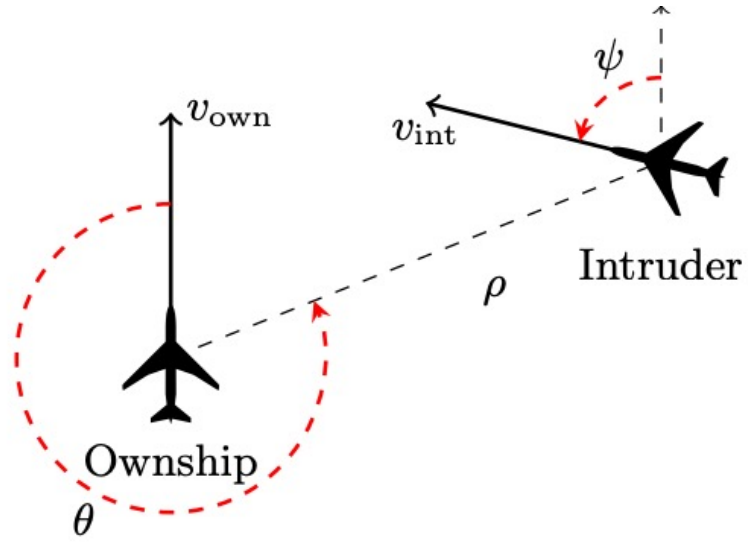
- <some condition> usually requires domain knowledge
- can only be specified *locally*
- Does not cover requirements that use the NN in closed-loop system

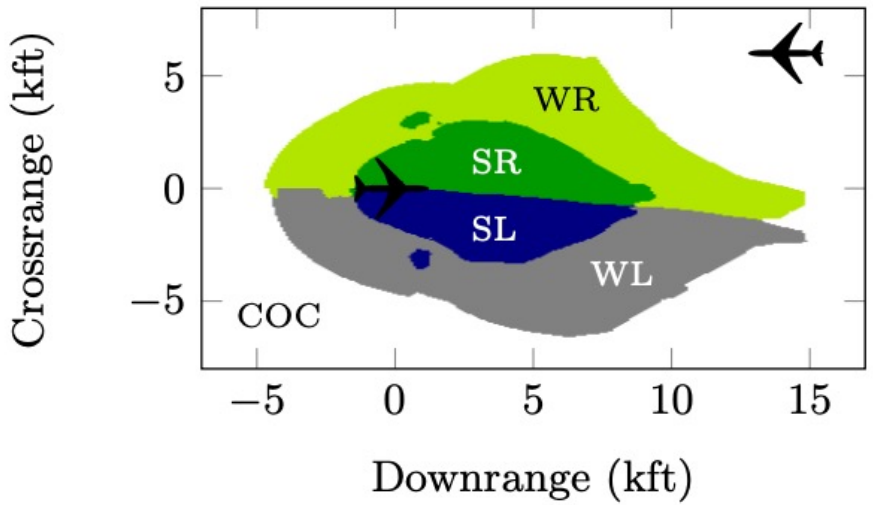


Application: ACAS Xu system

- 3MB DNN represents a large (2GB) lookup table for collision avoidance of unmanned aircraft
- Input: $\boldsymbol{x} \in \mathbb{R}^7$ ρ: Distance; θ: relative angle; ψ: relative heading; v_{own}, v_{in}: Speeds, τ :Time until loss of vertical separation; a_{prev}: Previous advisory.
- Output: Clear of Conflict (COC) or advisor weak/strong left/right.
- Network: 45 networks produced by discretizing τ and a_{prev}, each with 5 inputs and 5 outputs
- Requirement: E.g. If the intruder is far then the score for COC should be above some threshold

  $\forall \boldsymbol{x} \in \mathbb{R}^7\ \boldsymbol{x}.d\ >\ 55947, \boldsymbol{x}.v_{own}\ >\ 1145, \boldsymbol{x}.v_{in} \leq 60 \Rightarrow f(\boldsymbol{x}) \geq 1500$





General formulation of correctness

$$(precondition \wedge neural\ network) \Rightarrow postcondition$$

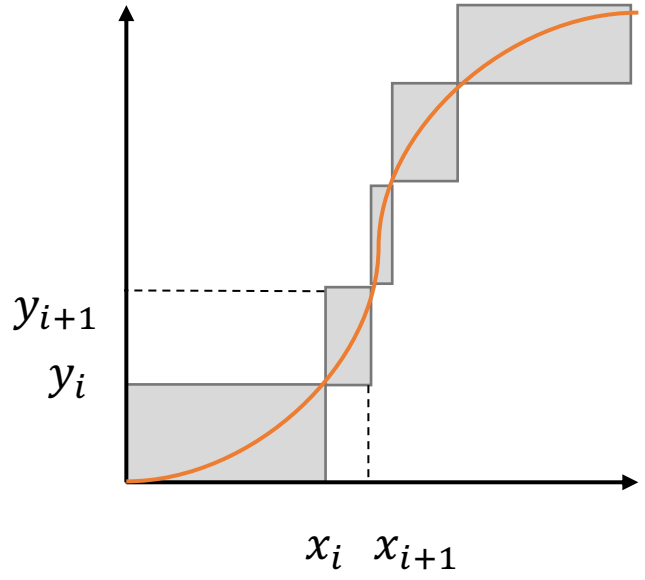
$$\left(\bigwedge_{i=1}^{n} |x_i - c_i| \leq 0.1\right) \wedge F_G \bigwedge\left(\bigwedge_{i=1}^{n} x_i = v_i^o\right) \wedge \left(\bigwedge_{i=k}^{k+m} r_i = v_i^o\right) \wedge \left(\bigwedge_{i=1}^{m} r_1 > r_{1+i}\right)$$

$$Precondition\ P$$
$$r_1 == \ f_{G1}(x_1)$$
$$r_2 == \ f_{G2}(x_2)$$
$$\ldots$$
$$r_l \ == f_{Gl}(x_l)$$
$$Postcondition\ Q$$

Precondition and Post-condition can be encoded in LRA. Solve using DPLL^{LRA}

Sigmoids and ReLUs and exponential blow-up

- Sigmoids can be approximated by piece-wise relations
- Others have considered MILP encodings
- Without disjunctions the problem is an LP
  - Polynomial Time solvable, Simplex
- If for all the inputs, the ReLUs are either active (x) or inactive (0)
  - We can replace each ReLU with either $x_i = f(x)$ or $x_i = 0$ based on lightweight analysis of which is which
- Otherwise, DPLL has to handle the disjunctions and in the worst case will have to consider every possible case (active or inactive) for every possible ReLU leading to exponential number of calls to Simplex
- Reluplex [Katz et al. 2017] addresses this problem



Reluplex Decision Procedure for ReLU NNs

Input: F in Reluplex form

Output: $\exists \boldsymbol{x} \in \mathbb{R}^m$ such that $\boldsymbol{x} \vDash F$?

- Delay case splitting on ReLUs
- In the worst case it is still exponential, but has been shown to be empirically better than DPLL$^{\text{LRA}}$

Reluplex Decision Procedure for ReLU NNs

Input: F in Reluplex form

Output: $\exists \boldsymbol{x} \in \mathbb{R}^m$ such that $\boldsymbol{x} \vDash F$?

Reluplex form

- Equations (same as Simplex)
- Bounds (same as Simplex)
- Relu: $x_i = relu(x_j)$

Given conjunction of inequalities and ReLU constraints we can convert them to Reluplex form and add $x_i \geq 0$.

Reluplex

Input: A formula F in Reluplex form

Output: $\boldsymbol{x} \vDash F$ or UNSAT

$\boldsymbol{x} \coloneqq \langle x_i \mapsto 0 \rangle$; F':= non ReLU part of F

while true **do**

$r \coloneqq Simplex(F', \boldsymbol{x})$

if r **is unsat then return UNSAT**

else if $r \vDash F$ **then return** r

// Handle violated ReLU constraints

Let $x_i = relu(x_j) \in F$ such that $\boldsymbol{x}[x_i \neq relu(\boldsymbol{x}[x_j)$

// Case split

If $u_j > 0, l_j < 0$ and $x_i = relu(x_j)$ considered > τ times

Reluplex

Input: A formula F in Reluplex form

Output: $\boldsymbol{x} \vDash F$ or UNSAT

$\boldsymbol{x} \coloneqq \langle x_i \mapsto 0 \rangle$; F':= non ReLU part of F

while true **do**

$r \coloneqq Simplex(F', \boldsymbol{x})$

if r **is unsat then return UNSAT**

else if $r \vDash F$ **then return** r

// Handle violated ReLU constraints

Let $x_i = relu(x_j) \in F$ such that $\boldsymbol{x}[x_i \neq relu(\boldsymbol{x}[x_j)$

if x_i is basic **then** pivot x_i with non-basic variable x_k where $k \neq j$ and $c_{ik} \neq 0$

if x_j is basic **then** pivot x_j with non-basic variable x_k where $k \neq i$ and $c_{jk} \neq 0$

$\boldsymbol{x}[x_i \coloneqq relu(\boldsymbol{x}[x_j)$ **OR** $\boldsymbol{x}[x_j \coloneqq \boldsymbol{x}[x_i$

// Case split

If $u_j > 0, l_j < 0$ and $x_i = relu(x_j)$ considered > τ times

$r_1 = Reluplex(F \wedge x_j \geq 0 \wedge x_j = x_j)$

$r_2 = Reluplex(F \wedge x_j \leq 0 \wedge x_i = 0)$

if $r_1 = r_2 =$ unsat then **return** UNSAT

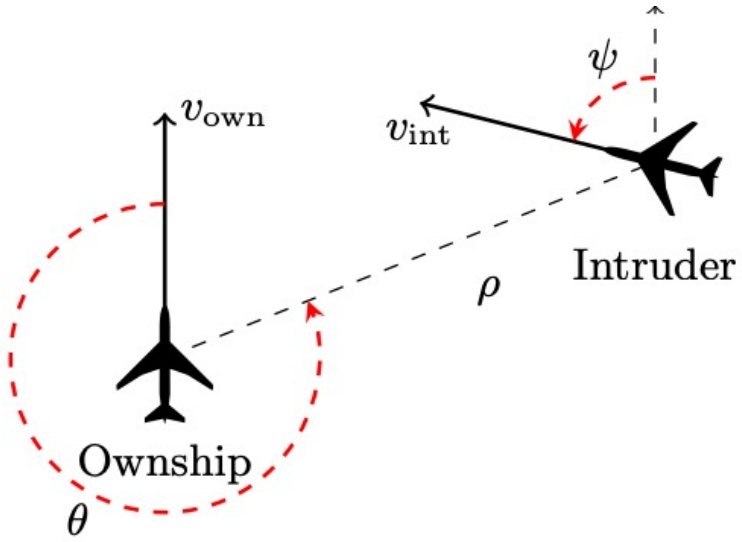
else if $r_1 = unsat$ **then return** r_1 **else return** r_2

Termination

- The last part of the algorithm ensures termination
- Otherwise, the algorithm could loop forever between fixing relu constraints and Simplex
- $F \equiv x_i = relu(x_j)$ is split into two cases
  - $F_1 \equiv \ \ x_j \geq 0 \wedge x_i = x_j$
  - $F_2 \equiv \ \ x_j \leq 0 \wedge x_i = 0$
  - $F' \equiv (F \wedge F_1) \vee (F \wedge F_2)$

Performance of Reluplex on ACAS

	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8
CVC4	-	-	-	-	-	-	-	-
Z3	-	-	-	-	-	-	-	-
Yices	1	37	-	-	-	-	-	-
MathSat	2040	9780	-	-	-	-	-	-
Gurobi	1	1	1	-	-	-	-	-
Reluplex	8	2	7	7	93	4	7	9



Summary

- Input-output and robustness requirements of neural networks can be expressed as satisfiability queries
- Without ReLU and disjunctions the requirements can be checked efficiently, e.g., Simplex
- Reluplex implements smarter case splitting