Neural Network Verification

Kumar Madhukar

Department of Computer Science and Engineering
Indian Institute of Technology Delhi

IITD Winter Systems School 2023

December 7, 2023

Reluplex (CAV 2017)

- based on the Simplex algorithm, extended to handle non-linear ReLU activation function
- motivation: DNNs are widely being used on real world problems, for doing complex tasks such as speech recognition, image classification, game playing, etc.
- safety and business-critical applications require formal guarantees
- verifying DNNs with ReLUs is NP-complete
- `https://arxiv.org/abs/1702.01135`

Simplex method

- algorithm for linear programming (a technique for optimizing a linear objective function, given some linear equalities and inequalities over reals)

- decision problem – general Simplex

- accepts two kinds of constraints: equalities, and (optionally) lower/upper bounds

- example:

$$x + y \geq 2 \ \wedge \ 2x - y \geq 0 \ \wedge \ -x + 2y \geq 1$$

$$x + y - s_1 = 0 \ \wedge \ 2x - y - s_2 = 0 \ \wedge \ -x + 2y - s_3 = 0$$
$$s_1 \geq 2 \ \wedge \ s_2 \geq 0 \ \wedge \ s_3 \geq 1$$

Example

	x	y
s_1	1	1
s_2	2	-1
s_3	-1	2

$$
\begin{aligned}
2 &\leq s_1 \\
0 &\leq s_2 \\
1 &\leq s_3
\end{aligned}
$$

Example

	s_1	y
x	1	-1
s_2	2	-3
s_3	-1	3

$$
\begin{aligned}
\alpha(x) &= 2 \\
\alpha(y) &= 0 \\
\alpha(s_1) &= 2 \\
\alpha(s_2) &= 4 \\
\alpha(s_3) &= -2
\end{aligned}
$$

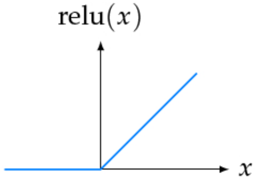
Example

	s_1	s_3
x	$2/3$	$-1/3$
s_2	1	-1
y	$1/3$	$1/3$

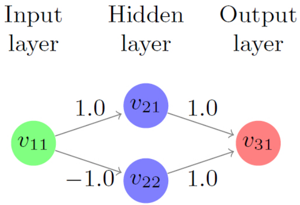
$$
\begin{aligned}
\alpha(x) &= 1 \\
\alpha(y) &= 1 \\
\alpha(s_1) &= 2 \\
\alpha(s_2) &= 1 \\
\alpha(s_3) &= 1
\end{aligned}
$$

Rectified Linear Units, ReLUs

- popular; piecewise linearity allows DNNs to generalize well
- $relu(x) = max(0, x)$

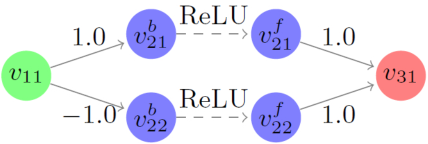


Toy example (from the introductory lecture)



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

ReLUs as variable pairs

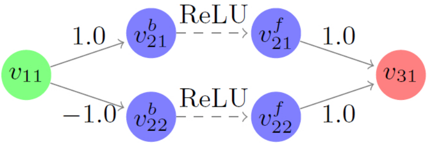


- *backward* (weighted sum) variables, and
  forward (activation function) variables

Backtracking algorithm, with case-splitting

- Linear programs (LP's) are easier to solve
- Piecewise linear constraints are reducible to LP's
- Backtracking algorithm:
    - fix each relu to *active* or *inactive* state
    - solve the resulting linear program
    - if a solution is found, we are done
    - otherwise, backtrack and try another option
- options exponential in the number of relu nodes

Reluplex



- extends the Simplex method
- does not require case-splitting in advance
- splitting only if necessary (heuristic based)

Revisiting Simplex

$$\text{Pivot}_1 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) < l(x_i), \quad x_j \in \text{slack}^+(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{Pivot}_2 \quad \frac{x_i \in \mathcal{B}, \quad \alpha(x_i) > u(x_i), \quad x_j \in \text{slack}^-(x_i)}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

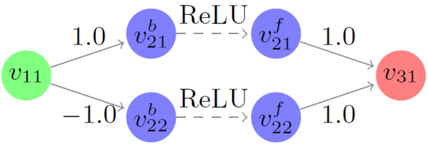
$$\text{Update} \quad \frac{x_j \notin \mathcal{B}, \quad \alpha(x_j) < l(x_j) \vee \alpha(x_j) > u(x_j), \quad l(x_j) \leq \alpha(x_j) + \delta \leq u(x_j)}{\alpha := update(\alpha, x_j, \delta)}$$

$$\text{Failure} \quad \frac{x_i \in \mathcal{B}, \quad (\alpha(x_i) < l(x_i) \ \wedge \ \text{slack}^+(x_i) = \emptyset) \vee (\alpha(x_i) > u(x_i) \ \wedge \ \text{slack}^-(x_i) = \emptyset)}{\texttt{UNSAT}}$$

$$\text{Success} \quad \frac{\forall x_i \in \mathcal{X}.\ l(x_i) \leq \alpha(x_i) \leq u(x_i)}{\texttt{SAT}}$$

$$\text{slack}^+(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} > 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} < 0 \wedge \alpha(x_j) > l(x_j))$$
$$\text{slack}^-(x_i) = \{x_j \notin \mathcal{B} \mid (T_{i,j} < 0 \wedge \alpha(x_j) < u(x_j)) \vee (T_{i,j} > 0 \wedge \alpha(x_j) > l(x_j))$$

Reluplex: example



Case-splitting

- not needed for this toy example, but can become necessary
- the same relu pair may keep breaking
- if that happens beyond a threshold, split
- solve the *active* and *inactive* cases separately
    - $(v_{ij}^b \geq 0) \land (v_{ij}^f = v_{ij}^b)$
    - $(v_{ij}^b < 0) \land (v_{ij}^f = 0)$
- if any of them reach a solution, it is done

Reluplex derivation rules

$$\text{Update}_b \quad \frac{x_i \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i)), \quad \alpha(x_j) \geq 0}{\alpha := update(\alpha, x_i, \alpha(x_j) - \alpha(x_i))}$$

$$\text{Update}_f \quad \frac{x_j \notin \mathcal{B}, \quad \langle x_i, x_j \rangle \in R, \quad \alpha(x_j) \neq \max(0, \alpha(x_i))}{\alpha := update(\alpha, x_j, \max(0, \alpha(x_i)) - \alpha(x_j))}$$

$$\text{PivotForRelu} \quad \frac{x_i \in \mathcal{B}, \quad \exists x_l.\ \langle x_i, x_l \rangle \in R \vee \langle x_l, x_i \rangle \in R, \quad x_j \notin \mathcal{B}, \quad T_{i,j} \neq 0}{T := pivot(T, i, j), \quad \mathcal{B} := \mathcal{B} \cup \{x_j\} \setminus \{x_i\}}$$

$$\text{ReluSplit} \quad \frac{\langle x_i, x_j \rangle \in R, \quad l(x_i) < 0, \quad u(x_i) > 0}{u(x_i) := 0 \qquad l(x_i) := 0}$$

$$\text{ReluSuccess} \quad \frac{\forall x \in \mathcal{X}.\ l(x) \leq \alpha(x) \leq u(x), \quad \forall \langle x^b, x^f \rangle \in R.\ \alpha(x^f) = \max(0, \alpha(x^b))}{\texttt{SAT}}$$

Reluplex algorithm: Efficient Implementation

- bound tightening (tighter upper/lower bounds can eliminate ReLUs)

$$\begin{aligned} x &= y + z \\ x &\geq -2 \\ y &\geq 1 \\ z &\geq 1 \end{aligned}$$

derives: $x \geq 2$

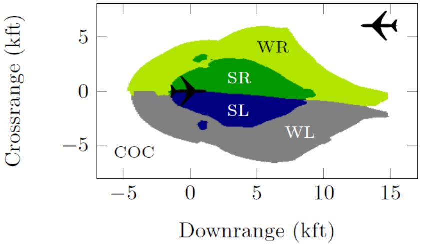
$$\text{deriveLowerBound} \quad \frac{x_i \in \mathcal{B}, \quad l(x_i) < \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot l(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot u(x_j)}{l(x_i) := \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot l(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot u(x_j)}$$

$$\text{deriveUpperBound} \quad \frac{x_i \in \mathcal{B}, \quad u(x_i) > \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot u(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot l(x_j)}{u(x_i) := \sum_{x_j \in \text{pos}(x_i)} T_{i,j} \cdot u(x_j) + \sum_{x_j \in \text{neg}(x_i)} T_{i,j} \cdot l(x_j)}$$

Reluplex algorithm: Efficient Implementation

- conflict analysis
  - bound tightening $\rightarrow$ contradictions ($lb.x > ub.x$) $\rightarrow$ backtracking
- floating-point arithmetic (round-off errors are kept small)
- standard way is to use precise computation (avoids round-off errors and ensures soundness)

Properties of interest



- no unnecessary turning advisories
- alterting regions are consistent, symmetric
- no strong alerts for large vertical separation

Properties: Example 1

- *if the intruder is near, and approaching from the left, the network advises strong right*
  - distance: $12000 \leq \rho \leq 62000$
  - angle to intruder: $0.2 \leq \theta \leq 0.4$

Properties: Example 2

- *if the vertical separation is large and the previous advisory is weak left, the network advises weak left*
    - time to loss of vertical separation, τ: 100
    - distance: $0 \leq \rho \leq 60760$
- counterexample found: 11 hours 8 minutes

Experiments

	Networks	Result	Time	Stack	Splits
ϕ_1	41	UNSAT	394517	47	1522384
	4	TIMEOUT			
ϕ_2	1	UNSAT	463	55	88388
	35	SAT	82419	44	284515
ϕ_3	42	UNSAT	28156	22	52080
ϕ_4	42	UNSAT	12475	21	23940
ϕ_5	1	UNSAT	19355	46	58914
ϕ_6	1	UNSAT	180288	50	548496
ϕ_7	1	TIMEOUT			
ϕ_8	1	SAT	40102	69	116697
ϕ_9	1	UNSAT	99634	48	227002
ϕ_{10}	1	UNSAT	19944	49	88520

Comparison to SMT and LP solvers

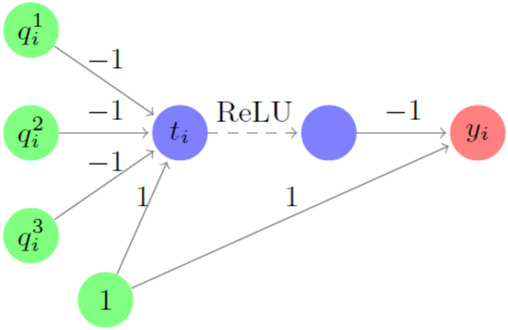
	φ_1	φ_2	φ_3	φ_4	φ_5	φ_6	φ_7	φ_8
CVC4	-	-	-	-	-	-	-	-
Z3	-	-	-	-	-	-	-	-
Yices	1	37	-	-	-	-	-	-
MathSat	2040	9780	-	-	-	-	-	-
Gurobi	1	1	1	-	-	-	-	-
Reluplex	8	2	7	7	93	4	7	9

- results on 2 networks, 8 simple properties, timeout $14400s$
- SMT solvers suffer from: precise arithmetic, lack of direct support for ReLUs
- Gurobi solved faster, but only instances that didn't need case-splitting

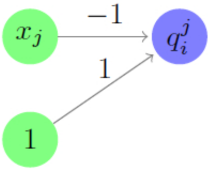
Complexity of the verification task

- Verifying properties in DNNs with ReLUs is NP-Complete.
  - DNN $\mathcal{N}$, property ϕ (conjunction of linear constraints on input and output)
  - ϕ is satisfiable on $\mathcal{N}$ if there exists an assignment α, such that $\alpha(\mathit{output}) = \mathcal{N}(\alpha(\mathit{input}))$, and α satisfies ϕ
- membership is easy; the witness can be simulated on $\mathcal{N}$
- for NP-hardness, we show that 3-SAT can be reduced to this

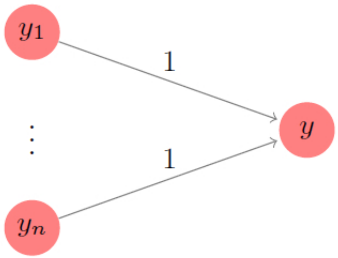
Disjunction gadget



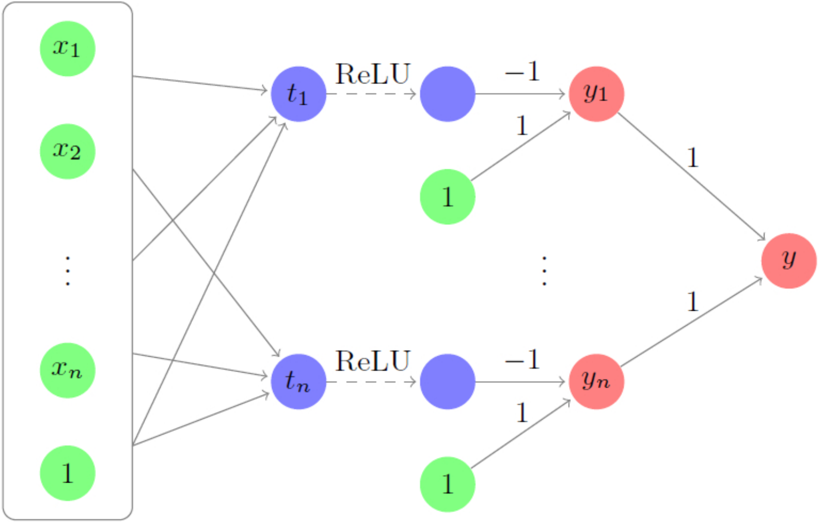
Negation gadget



Conjunction gadget



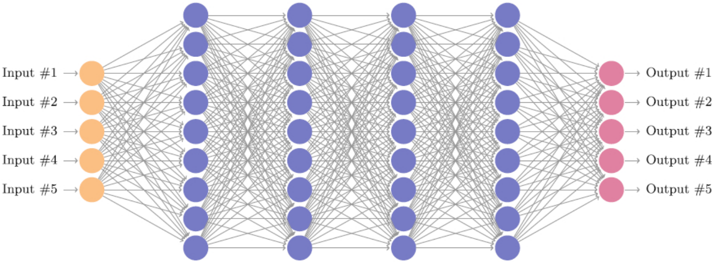
Reduction



Abstraction-Refinement (CAV 2020) A well-known story in formal verification

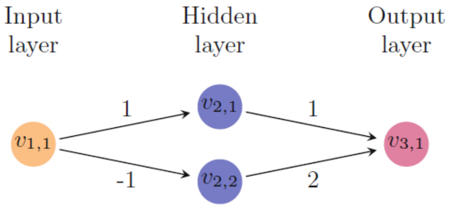
- replace the DNN $\mathcal{N}$ by a "smaller" (*abstract*) network $\overline{\mathcal{N}}$
- verify $\overline{\mathcal{N}}$; by construction, if $\overline{\mathcal{N}}$ meets the spec, so does $\mathcal{N}$
- if $\overline{\mathcal{N}}$ fails to meet the spec, there must be counterexample x
- if x is actual, $\mathcal{N}$ violates the spec
- else refine $\overline{\mathcal{N}}$ (little more accurate, and "larger")
- done using the spurious x
  (Counterexample-Guided Abstraction Refinement, or CEGAR)
- `https://arxiv.org/abs/1910.14574`

Background: Neural Networks



- feedforward neural network (missing: edge weights and activation function)
- evaluate a neuron: compute weighted sum, and apply activation function
- $\mathrm{ReLU}(x) = \max(0, x)$, called Rectified Linear Unit

An example

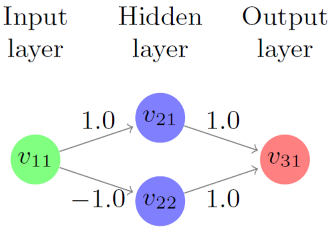


- three layers; input $v_{1,1}$ is 3
- node $v_{2,1}$ evaluates to 3, and node $v_{2,2}$ evaluates to 0
- output node $v_{3,1}$ evaluates to 3

Verification

- precondition $\mathcal{P}$, postcondition $\mathcal{Q}$, network $\mathcal{N}$
- is there an input x that satisfies $\mathcal{P}(x)$ and $\mathcal{Q}(y)$, where $y = \mathcal{N}(x)$
- assumptions made in this paper:
  - (on $\mathcal{N}$) - only ReLU activation functions; single output node
  - (on $\mathcal{P}$) - conjunctions of linear constraints on input values
  - (on $\mathcal{Q}$) - $y > c$, for a given constant c
- not as limiting as it may seem (let us come back to this in the end)

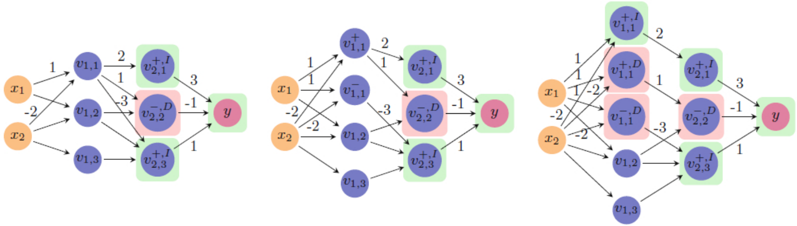
Recall the toy example



Abstraction

- transform the neural network $\mathcal{N}$ into $\overline{\mathcal{N}}$, such that $\mathcal{N}(x) \leq \overline{\mathcal{N}}(x)$, for every input x
- if abstract is safe ($\overline{\mathcal{N}}(x) \leq c$), then so is the concrete ($\mathcal{N}(x) \leq c$)
- abstraction-refinement: merging neurons (and then splitting back)
- but not on $\mathcal{N}$ (on an equivalent network $\mathcal{N}''$)

$\mathcal{N} \to \mathcal{N}' \to \mathcal{N}''$ (all equivalent)

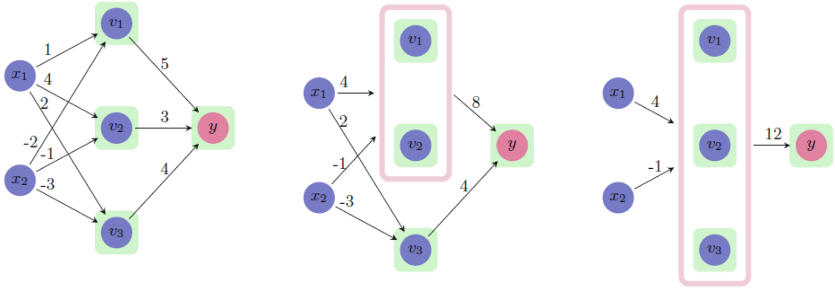


- every hidden neuron should either be `pos` or `neg`
- based on weights of outgoing edges; split if needed ($\mathcal{N}'$)
- also, every neuron must be `inc` or `dec`; split if needed
- depending on whether increasing (or decreasing) its value results in an increased output (traversing backwards)

The abstract operator

- merges a pair of neurons; can be done multiple times
- merge only if the `pos`/`neg` and `inc`/`dec` attributes are same
- for the (`pos, inc`) and (`neg, inc`) case
  - take max of incoming, and sum of outgoing
- for the (`pos, dec`) and (`neg, dec`) case
  - take min of incoming, and sum of outgoing
- intuitively, the new node contributes more to the output (than the two original nodes)

An example



$$y = 5R(x_1 - 2x_2) + 3R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$$

$$y = 8R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$$

$$y = 12R(4x_1 - x_2)$$

- abstraction is independent of the order in which it was done

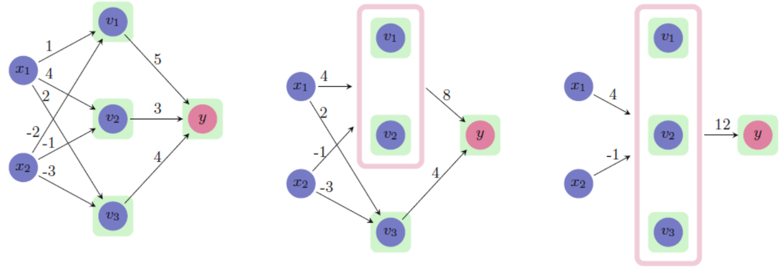
The need to refine

- of course, if the abstraction is too coarse
- suppose $\mathcal{N}(x_0) = 3$, $\overline{\mathcal{N}}(x_0) = 8$, and the property is $\overline{\mathcal{N}}(x) \leq 6$
- need to refine $\overline{\mathcal{N}}$ into $\overline{\mathcal{N}}'$, such that for every x, $\mathcal{N}(x) \leq \overline{\mathcal{N}}'(x) \leq \overline{\mathcal{N}}(x)$
- refine picks a concrete neuron from an abstract neuron, and puts it back in the network

More about the abstraction

- apply *abstraction to saturation* (to at most 4 neurons in every hidden layer)
- can be controlled based on certain heuristics
- inaccuracies by caused by the max and min operators
- merge neurons that approximate least; split one that restores the most

Merging heuristics

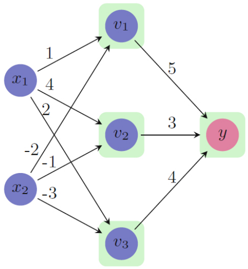


$y = 5R(x_1 - 2x_2) + 3R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$

$y = 8R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$

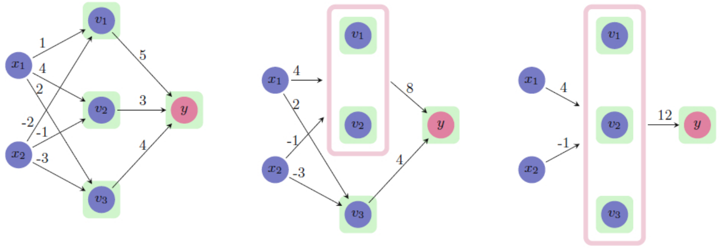
$y = 12R(4x_1 - x_2)$

- merge: maximal value of $|a - b|$ (over all incoming edges with weights a and b) is minimal
- the new edge is "closest" to the replaced ones (saving a neuron anyway!)



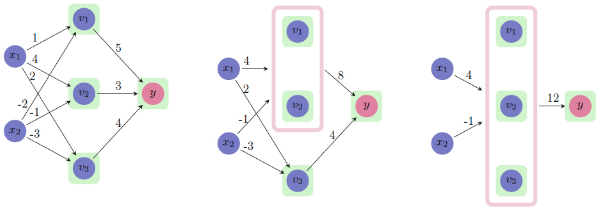
- merging (v_1, v_2), the (a, b) pairs are: (1,4), (-2, -1)
- $\max(\,|1-4|\,, \left|-2-(-1)\right|\,) = 3$
- merging (v_1, v_3), the (a, b) pairs are: (1,2), (-2, -3)
- $\max(\,|1-2|\,, \left|-2-(-3)\right|\,) = 1$
- merging (v_2, v_3), the (a, b) pairs are: (4,2), (-1, -3)
- $\max(\,|1-2|\,, \left|-2-(-3)\right|\,) = 2$
- merge (v_1, v_3) first

Splitting heuristics



$y = 5R(x_1 - 2x_2) + 3R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$ $\quad$ $y = 8R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$ $\quad$ $y = 12R(4x_1 - x_2)$

- split: v from $\overline{v}$, by considering
  - edge-weight difference between v and $\overline{v}$
  - difference between $v(x)$ and $\overline{v}(x)$, for the counterexample x



$y = 5R(x_1 - 2x_2) + 3R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$ $\quad$ $y = 8R(4x_1 - x_2) + 4R(2x_1 - 3x_2)$ $\quad$ $y = 12R(4x_1 - x_2)$

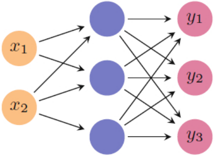
- consider the counterexample $(x_1 = 1, x_2 = 0)$
- original neurons' evaluation: $(v_1 = 1, v_2 = 4, v_3 = 2)$
- abstract neuron's evaluation: $(\overline{v} = 4)$
- wt. diff. (between v_1 and $\overline{v}$) for in-edge from x_1, x_2: 3, 1
- wt. diff. (between v_2 and $\overline{v}$) for in-edge from x_1, x_2: 0, 0
- wt. diff. (between v_3 and $\overline{v}$) for in-edge from x_1, x_2: 2, 2
- remove v_1, (wt. diff $*$ val. diff.) is largest: (9, 0, 4)

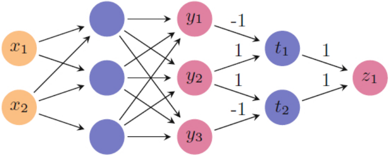
The complete CEGAR algorithm

Algorithm 1. Abstraction-based DNN Verification(N, P, Q)

1: Use `abstract` to generate an initial over-approximation $\bar{N}$ of N
2: **if** *Verify*($\bar{N}, P, Q$) is `UNSAT` **then**
3:     return `UNSAT`
4: **else**
5:     Extract counterexample c
6:     **if** c is a counterexample for N **then**
7:         return `SAT`
8:     **else**
9:         Use `refine` to refine $\bar{N}$ into $\bar{N}'$
10:         $\bar{N} \leftarrow \bar{N}'$
11:         Goto step 2
12:     **end if**
13: **end if**

Reducing a complex property (in the desired form)





- consider the property $(y_2 > y_1) \vee (y_2 > y_3)$
- encoded by adding neurons t_1, t_2, and z_1
- $t_1 = \max(0, y_2 - y_1)$
- $t_2 = \max(0, y_2 - y_3)$
- $z_1 = t_1 + t_2$
- property: $z_1 > 0$ (*iff* $t_1 > 0 \vee t_2 > 0$)

Experiments

- 45 DNNs from ACAS
- input is a set of sensor readings (speed, direction, location, etc.)
- five output neurons - possible turning advisories (left, right, clear-of-conflict, etc.)
- each DNN has 300 hidden neurons, across 6 hidden layers (leading to 1200 neurons after the transformation)

Findings

- abstraction to saturation outperforms indicator-guided abstraction
- avg. 269 nodes were needed to prove (the original has 310)
- "simpler" queries may sometimes be better than smaller networks
- reconfirmed in another set of experiments: even though network size increased (to avg. 385, from 310), abstracted versions were easier to verify that the original
- even further reduction on adversarial robustness properties

Modifying DNNs (LPAR 2020): Motivation

- change an existing DNN in a “small” way
- DNNs may have a bug (an undesirable behavior) that needs fixing
- should not impact the other functionality significantly
- one may retrain, but it is expensive and may lead to a very different DNN

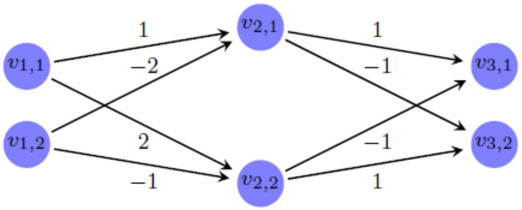
DNN Verification Problem

- network N, precondition P, and postcondition Q
- does there exist an input x such that $P(x)$ and $Q(y)$ hold, where $y = N(x)$

DNN Modification Problem

Definition 1. ***The DNN Modification Problem.*** *Let N denote a DNN, let X denote a set of fixed input points $X = \{x_1, \ldots, x_n\}$, and let Q denote a predicate over the classifications $N(x_1), \ldots, N(x_n)$ of the points of X. The* DNN modification problem *is to find a new DNN, N', such that $Q(N'(x_1), \ldots, N'(x_n))$ holds, and such that the distance between N and N' is at most some $\delta > 0$.*

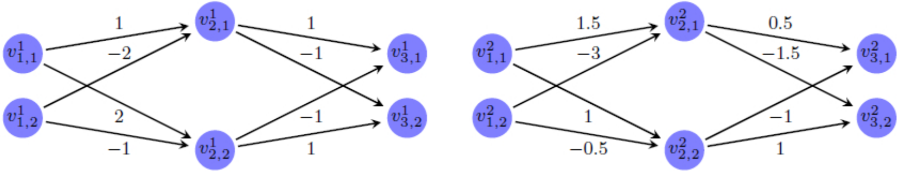
Example



- let $X = \{\langle 3, 4\rangle\}$ and $Q(N'(\langle 3, 4\rangle)) = v_{3,1} \geq v_{3,2}$

DNN Distance

- DNNs of identical topology



(a) N^1 (b) N^2

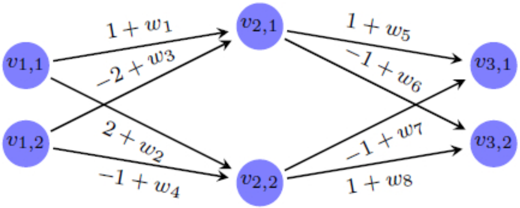
$$\left\|N^1 - N^2\right\|_1 = (|-0.5| + |1| + |1| + |-0.5| + |0.5| + |0.5| + 0 + 0) = 4$$
$$\left\|N^1 - N^2\right\|_\infty = max\left\{|-0.5|, |1|, |1|, |-0.5|, |0.5|, |0.5|, 0, 0\right\} = 1$$

Minimal Modification

- find the closest N' that solves the DNN Modification Problem
- repeatedly solving the modification problem as part of a binary search
- optimization problem, but highly non-convex and high-dimensional

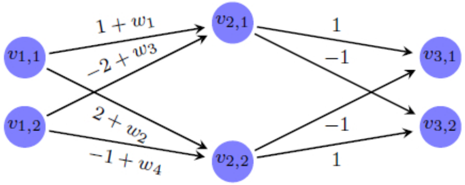
Example

- let $X = \{\langle 3,4\rangle\}$ and $Q(N'(\langle 3,4\rangle)) = v_{3,1} \geq v_{3,2}$



$$v_{3,1} = (1+w_5)\cdot \mathrm{ReLU}(3(1+w_1)+4(-2+w_3)) + (-1+w_7)\cdot \mathrm{ReLU}(3(2+w_2)+4(-1+w_4))$$
$$v_{3,2} = (-1+w_6)\cdot \mathrm{ReLU}(3(1+w_1)+4(-2+w_3)) + (1+w_8)\cdot \mathrm{ReLU}(3(2+w_2)+4(-1+w_4))$$

Single layer modification

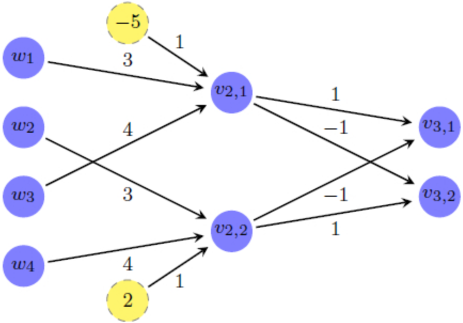


$$
\begin{aligned}
v_{3,1} &= \mathrm{ReLU}(3(1+w_1) + 4(-2+w_3)) - \mathrm{ReLU}(3(2+w_2) + 4(-1+w_4)) \\
&= \mathrm{ReLU}(3w_1 + 4w_3 - 5) - \mathrm{ReLU}(3w_2 + 4w_4 + 2) \\
v_{3,2} &= -\mathrm{ReLU}(3(1+w_1) + 4(-2+w_3)) + \mathrm{ReLU}(3(2+w_2) + 4(-1+w_4)) \\
&= -\mathrm{ReLU}(3w_1 + 4w_3 - 5) + \mathrm{ReLU}(3w_2 + 4w_4 + 2)
\end{aligned}
$$

Single layer modification as DNN verification

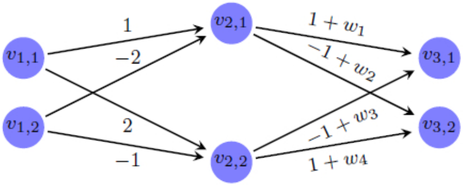
$$
\begin{aligned}
v_{3,1} = &\ \mathrm{ReLU}(3(1+w_1)+4(-2+w_3)) - \mathrm{ReLU}(3(2+w_2)+4(-1+w_4)) \\
= &\ \mathrm{ReLU}(3w_1+4w_3-5) - \mathrm{ReLU}(3w_2+4w_4+2) \\
v_{3,2} = &-\mathrm{ReLU}(3(1+w_1)+4(-2+w_3)) + \mathrm{ReLU}(3(2+w_2)+4(-1+w_4)) \\
= &-\mathrm{ReLU}(3w_1+4w_3-5) + \mathrm{ReLU}(3w_2+4w_4+2)
\end{aligned}
$$

$$P = \bigwedge_{i=1}^{4} -\delta \leq w_i \leq \delta$$



$$Q = v_{3,1} \geq v_{3,2}$$

Output layer modification



$$
\begin{aligned}
\text{Minimize :} \quad & M \\
\text{Subject to :} \quad & M \geq 0 \\
& -M \leq w_1 \leq M \\
& -M \leq w_2 \leq M \\
& -M \leq w_3 \leq M \\
& -M \leq w_4 \leq M \\
& v_{3,1} = 0 \cdot (1 + w_1) + 2 \cdot (-1 + w_3) \\
& v_{3,2} = 0 \cdot (-1 + w_2) + 2 \cdot (1 + w_4) \\
& v_{3,1} \geq v_{3,2}
\end{aligned}
$$

Thank you!