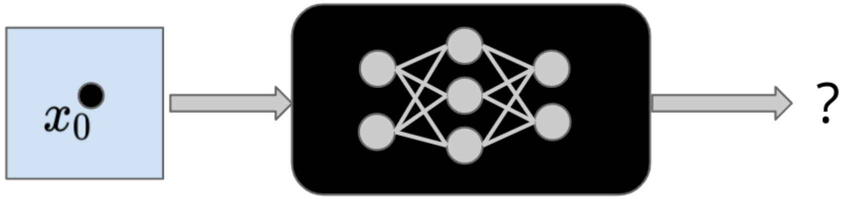
Introduction to
Neural Network Verification

Book Written by Prof. Aws Albarghouthi
UW-Madison

Presented by Yang Yang and Klaus Peng
2023/10/11

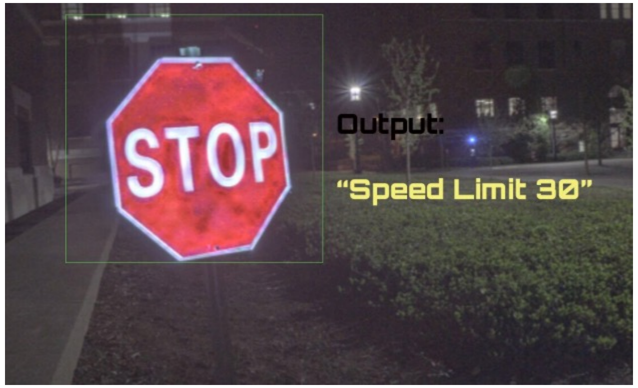
Neural Network Verification



Neural network (f) works as a black box. $f : R^n \rightarrow R^m$

Challenge

The network could behave **unexpectedly** and produce **wrong** results.



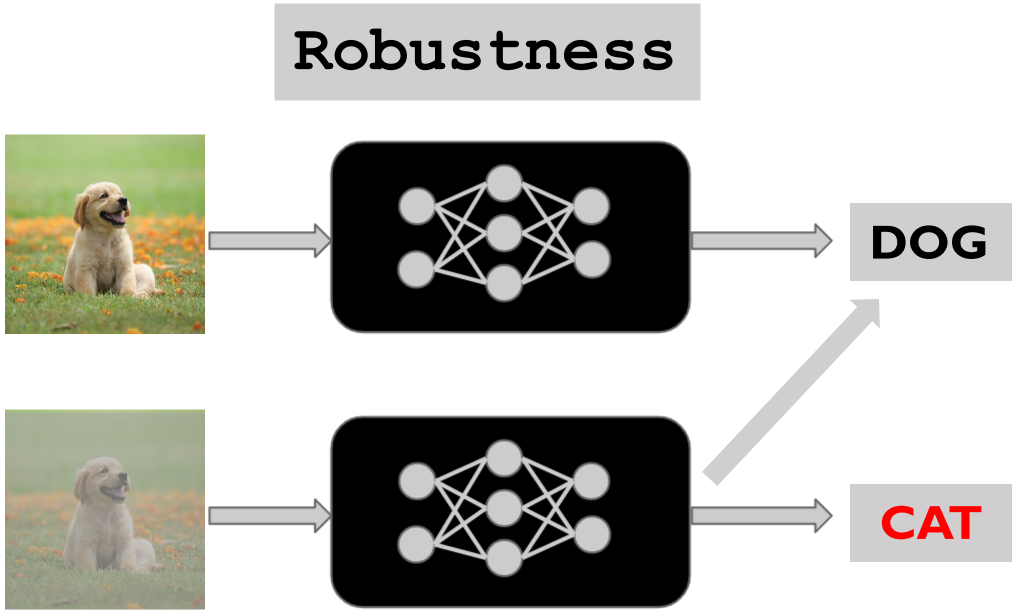
Goal

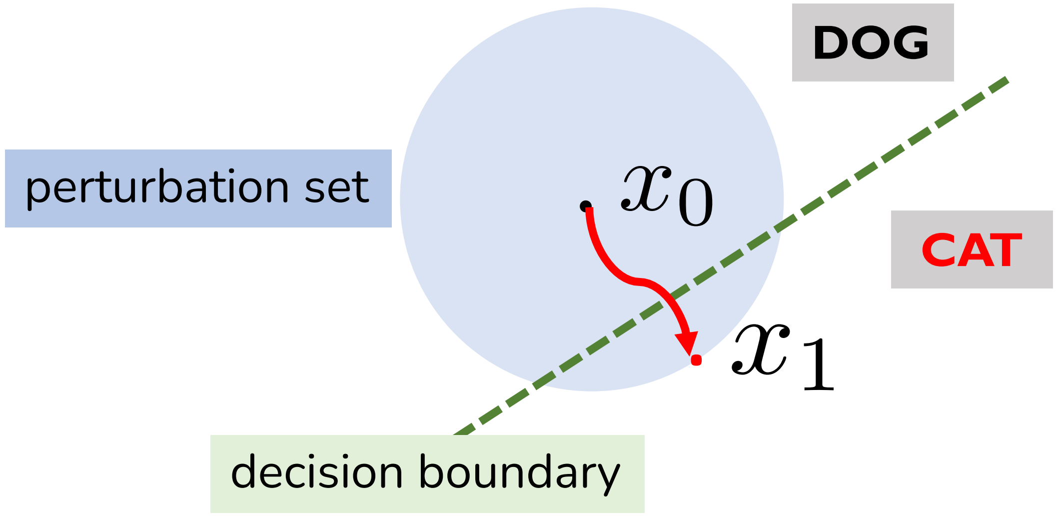
We want to make **NN**
have some desired **properties**
we can **formally trust**.

Neural Network Verification

Neural Network Properties

We want to make **NN** have some desired **properties** we can **formally trust**.





$f : R^n \to R^m$ →

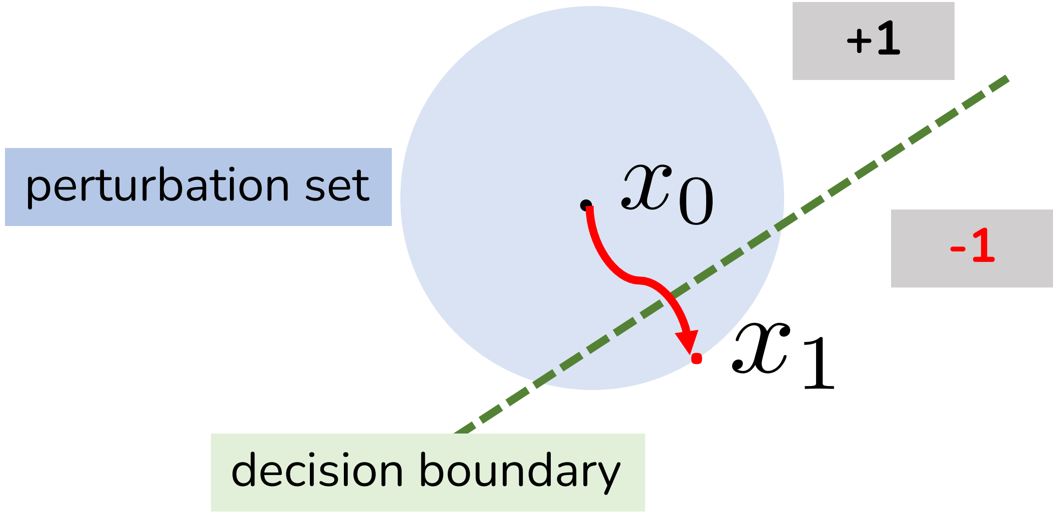
$\{\ \lvert x - c \rvert \leqslant 0.1\ \}$	{precondition}
$r_1 \leftarrow f(x)$ $r_2 \leftarrow f(c)$	{function calls}
$\{\ \text{class}(r_1) = \text{class}(r_2)\ \}$	{postcondition}

→

For any values of $x_1, \ldots, x_n$ ***that*** make the **precondition** **true**, let $r_1 = f(x_1)$, $r_2 = g(x_2)$, ... ***Then*** the **postcondition** is **true**.

$$pre \land \mathbf{F} \Rightarrow post$$

Neural Network Verification



Assuming $f(x_0) > 0$, we solve the optimization problem to find the worst case:

$$f^* = \min \{ f(x) \}, x \in P$$

Where P is usually a perturbation set on x_0, e.g.,

$$P = \{ x \mid \|x - x_0\|_p \leq \varepsilon \}$$

$f^* > 0$

Challenge

In real-world case, non-linear, non-convex constraints and generally leads to **NP-complete**.

Review

Goal

We want to make **NN**
have some desired **properties** we can **formally trust**.

$f : R^n \to R^m$ →

$\{ \, \lvert x - c \rvert \leqslant 0.1 \, \}$	{precondition}
$r_1 \leftarrow f(x)$ $r_2 \leftarrow f(c)$	{function calls}
$\{ \, \text{class}(r_1) = \text{class}(r_2) \, \}$	{postcondition}

→ ***For any values*** of $x_1, \ldots, x_n$ ***that*** make the **precondition** **true**, let $r_1 = f(x_1), r_2 = g(x_2), \ldots$. ***Then*** the **postcondition** is **true**.

$pre \wedge F \Rightarrow post$

Challenge

In real-world case, non-linear, non-convex constraints and generally leads to **NP-complete**.

NEXT?

Precondition, postcondition, true, false, **NP-complete**, ... → **SAT**

$pre \wedge F \Rightarrow post$ **Boolean Logic**

Propositional Logic Recap

Proposition is a **sentence** that could be either **true** or **false**.

A: SE4ML is an interesting course. **(X)**
B: SE4ML is a graduate-level course. **(√)**

Formula is a **string** that could be **generated** using proposition.

$$\mathbf{F} := \mathbf{F} \wedge \mathbf{F} \mid \mathbf{F} \vee \mathbf{F} \mid \neg \mathbf{F}$$
$$\mathbf{F} \triangleq (p \wedge q) \vee \neg r$$

Implication: $\mathbf{A} \Longrightarrow \mathbf{B} := \neg \mathbf{A} \vee \mathbf{B}$ $\qquad$ $pre \wedge \mathbf{F} \Longrightarrow post$

Interpretation is a **set** that assigns true or false to fv's elements.

$$fv(\mathbf{F}) = \{p, q, r\}$$
$$\mathbf{I} = \{ p \longmapsto \mathbf{true},\ q \longmapsto \mathbf{true},\ r \longmapsto \mathbf{false}\}$$

Formula $\mathbf{F}$ is <u>satisfiable (SAT)</u> if there **exists** an interpretation $\mathbf{I}$ such that $\mathbf{eval}(\mathbf{I}(\mathbf{F}))=\mathbf{true}$.

P: **Solve** the problem in polynomial time, bubble sort.
NP: **Validate** a solution in polynomial time, TSP.
NP-Complete: An **<u>NP problem</u>** that every NP problem is **<u>reducible to it</u>**, The Satisfiability Problem(**SAT**).

SAT: $\forall n_1, \ldots, nN$, whether $\mathbf{eval}(\mathbf{F}(n_1,\ldots,n_N))=\mathbf{true}$ $\qquad$ $\mathbf{F} := \mathbf{A} \vee \neg A$

How you solve it?

[STOC 1971, Stephen Cook] **The Complexity of Theorem-Proving Procedures**

Proved SAT to be the first NP-Complete using Turing Machine. $\Longrightarrow$ **1982 ACM Turing Award**

Arithmetic Logic

Satisfiability Modulo Theories (SMT)

$$F \triangleq p \lor q,\ fv(F) = \{p, q\}$$

Boolean Variable

$$p \leftarrow a + b \leq 10$$
$$q \leftarrow c = d[1]$$
$$F' \triangleq a + b \leq 10 \lor c = d[1]$$
$$fv(F') = \{a, b, c, d\}$$

Boolean Expression

Linear Real Arithmetic (LRA)

$$\sum_{i=1}^{n} c_i x_i + b \leq 0 \text{ or } \sum_{i=1}^{n} c_i x_i + b < 0$$

Linear Inequality

Non-Linear Arithmetic

Arithmetic Logic

SMT Solvers	*vs.*	MILP Solver

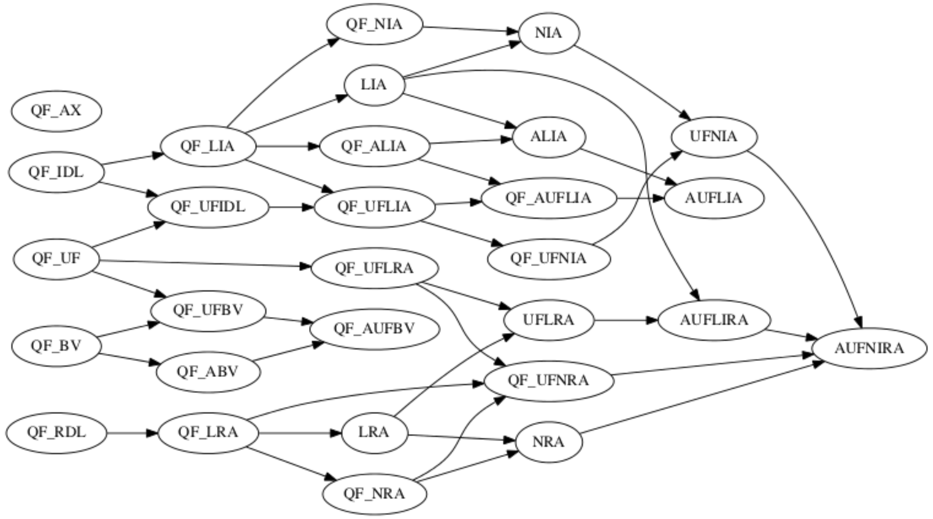


Figure 1: Overview of logics in SMT-LIB

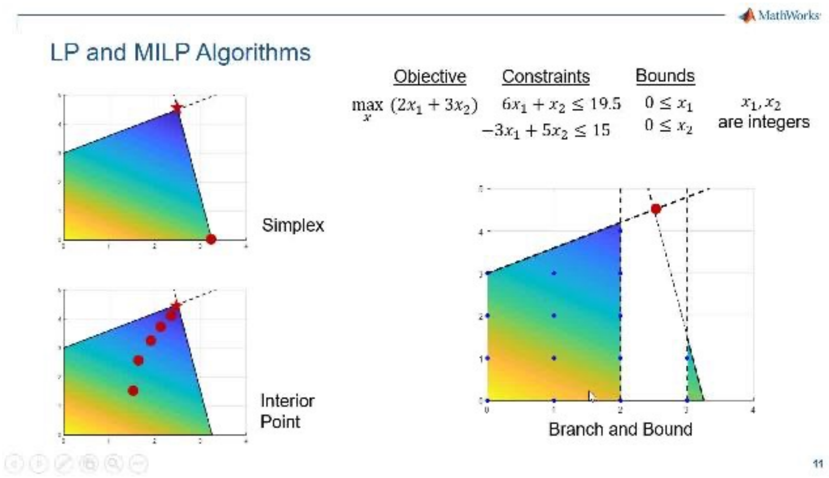


Figure 2: provided by Matlab

Advantages of SMT Solvers

- Allows encoding using **various** theories/logics
- **Ability** to combine multiple theories

Advantages of MILP Solver

- Sometimes, **faster** compared to SMT solver

Review

We have tools **how** it could be used in NNV? Why we need such **logic**?

Boolean Variable

Satisfiability Modulo Theories (SMT) — *Boolean Expression*

Linear Real Arithmetic (LRA) — *Linear Inequality*

$$pre \land \mathbf{F} \Longrightarrow post$$

SAT: $\forall n_1, \ldots, nN$, whether **eval**$(\mathbf{F}(n_1,\ldots,n_N))=$**true**

Use SMT to define the **specifications** of our neural network

Apply SMT solvers to **verify** if our model is robust or of other properties

NEXT? **Let's verify the real Neural Network !**

Encoding Nodes

Example of Nodes:

$$f_v(x) = 2x + 1$$

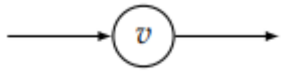
$$\downarrow$$

$$F_v \triangleq v^o = 2x + 1$$

$$f_v(x, y) = 3x - 2y$$

$$\downarrow$$

$$F_v \triangleq v^o = 3x - 2y$$



$$f_v(x) = relu(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

$$\downarrow$$

$$F_v \triangleq (x > 0 \Rightarrow v^o = x) \wedge (x \leq 0 \Rightarrow v^o = 0)$$

Formalized definition:

$$f_v(\mathbf{x}) = \begin{cases} \sum_{j=1}^{n_v} c_j^1 \cdot x_j + b^1 & \text{if } S_1 \\ \vdots & \vdots \\ \sum_{j=1}^{n_v} c_j^l \cdot x_j + b^l & \text{if } S_l \end{cases} \Rightarrow F_v \triangleq \bigwedge_{i=1}^{l} [S_i \Rightarrow (v^o = \sum_{j=1}^{n_v} c_j^i \cdot v^{in,j} + b^i)]$$

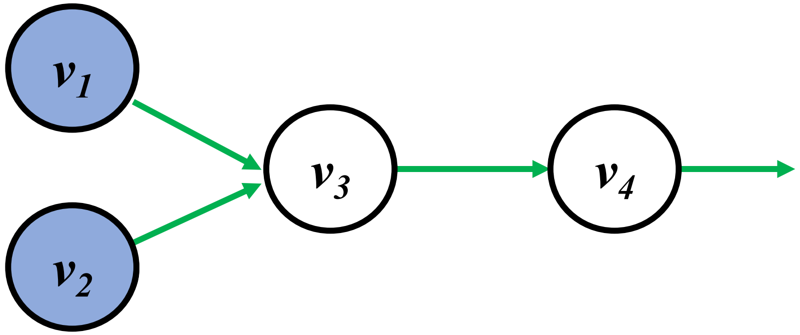
v^o - output of node v

$v^{in,i}$ - ith input to node v

Encoding Neural Network

1. Encoding Semantics of Nodes

$$F_V \triangleq \bigwedge_{v \in V \setminus V^{in}} F_v$$



$$V^{in} = \{v_1, v_2\},\ V^o = \{v_4\},\ f_{v_3} = 2x_1 - 3x_2,\ f_{v_4} = relu(x)$$

Formula for each non-input nodes:

$$F_{v_3} \triangleq v_3^o = 2v_3^{in,1} - 3v_3^{in,2}$$
$$F_{v_4} \triangleq (v_4^{in,1} > 0 \Rightarrow v_4^o = v_4^{in,1}) \wedge (v_4^{in,1} \leq 0 \Rightarrow v_4^o = 0)$$

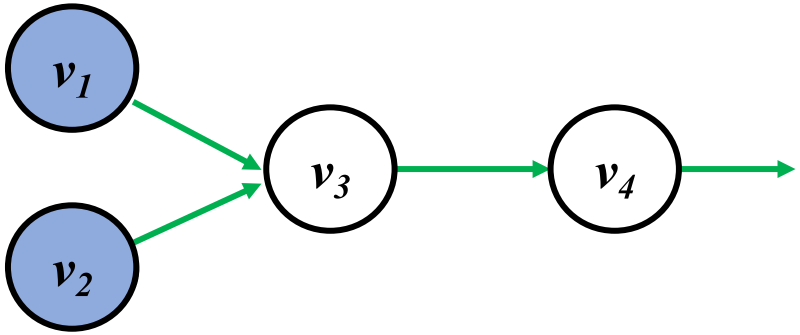
Put them together: $F_V \triangleq F_{v_3} \wedge F_{v_4}$

Encoding Neural Network

2. Encoding Edges Between Nodes

$$F_{o\to v} \triangleq \bigwedge_{i=1}^{n} v^{in,i} = v_j^o$$

$$F_E \triangleq \bigwedge_{v\in V\setminus V^{in}} F_{o\to v}$$



$V^{in} = \{v_1, v_2\},\ V^o = \{v_4\},\ f_{v_3} = 2x_1 - 3x_2,\ f_{v_4} = relu(x)$

Formula for each edge:

$$F_{o\to v_3} \triangleq (v_3^{in,1} = v_1^o) \wedge (v_3^{in,2} = v_2^o)$$
$$F_{o\to v_4} \triangleq v_4^{in,1} = v_3^o$$

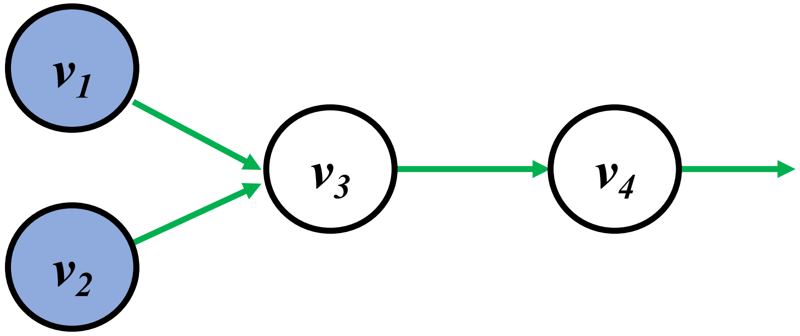
Put them together:

$$F_E \triangleq F_{o\to v_3} \wedge F_{o\to v_4}$$

Encoding Neural Network

3. All Together

$$F_G \triangleq F_V \wedge F_E$$



$$V^{in} = \{v_1, v_2\},\ V^o = \{v_4\},\ f_{v_3} = 2x_1 - 3x_2,\ f_{v_4} = relu(x)$$

Node & Edge:

$$F_{v_3} \triangleq v_3^o = 2v_3^{in,1} - 3v_3^{in,2}$$
$$F_{v_4} \triangleq (v_4^{in,1} > 0 \Rightarrow v_4^o = v_4^{in,1}) \wedge (v_4^{in,1} \leq 0 \Rightarrow v_4^o = 0)$$

$$F_V \triangleq F_{v_3} \wedge F_{v_4}$$

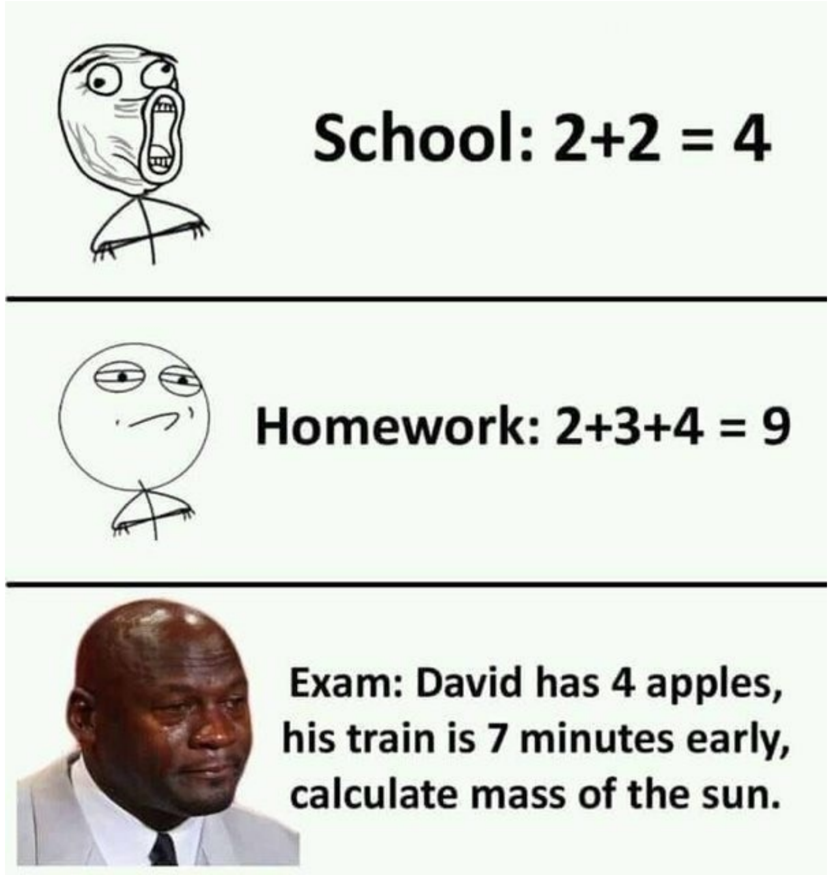
$$F_{o \to v_3} \triangleq (v_3^{in,1} = v_1^o) \wedge (v_3^{in,2} = v_2^o)$$
$$F_{o \to v_4} \triangleq v_4^{in,1} = v_3^o$$

$$F_E \triangleq F_{o \to v_3} \wedge F_{o \to v_4}$$

Finally:

$$F_G \triangleq F_{v_3} \wedge F_{v_4} \wedge F_{o \to v_3} \wedge F_{o \to v_4}$$

Encoding Correctness Properties



Recall previously:

$$\textit{pre} \wedge \mathbf{F} \Longrightarrow \textit{post}$$

$$\{\ |x - c| \leqslant 0.1\ \}$$
$$r_1 \leftarrow f(x)$$
$$r_2 \leftarrow f(c)$$
$$\{\ \text{class}(r_1) = \text{class}(r_2)\ \}$$

$$\left(P \wedge \bigwedge_{i=1}^{l} F_i\right) \Rightarrow Q$$

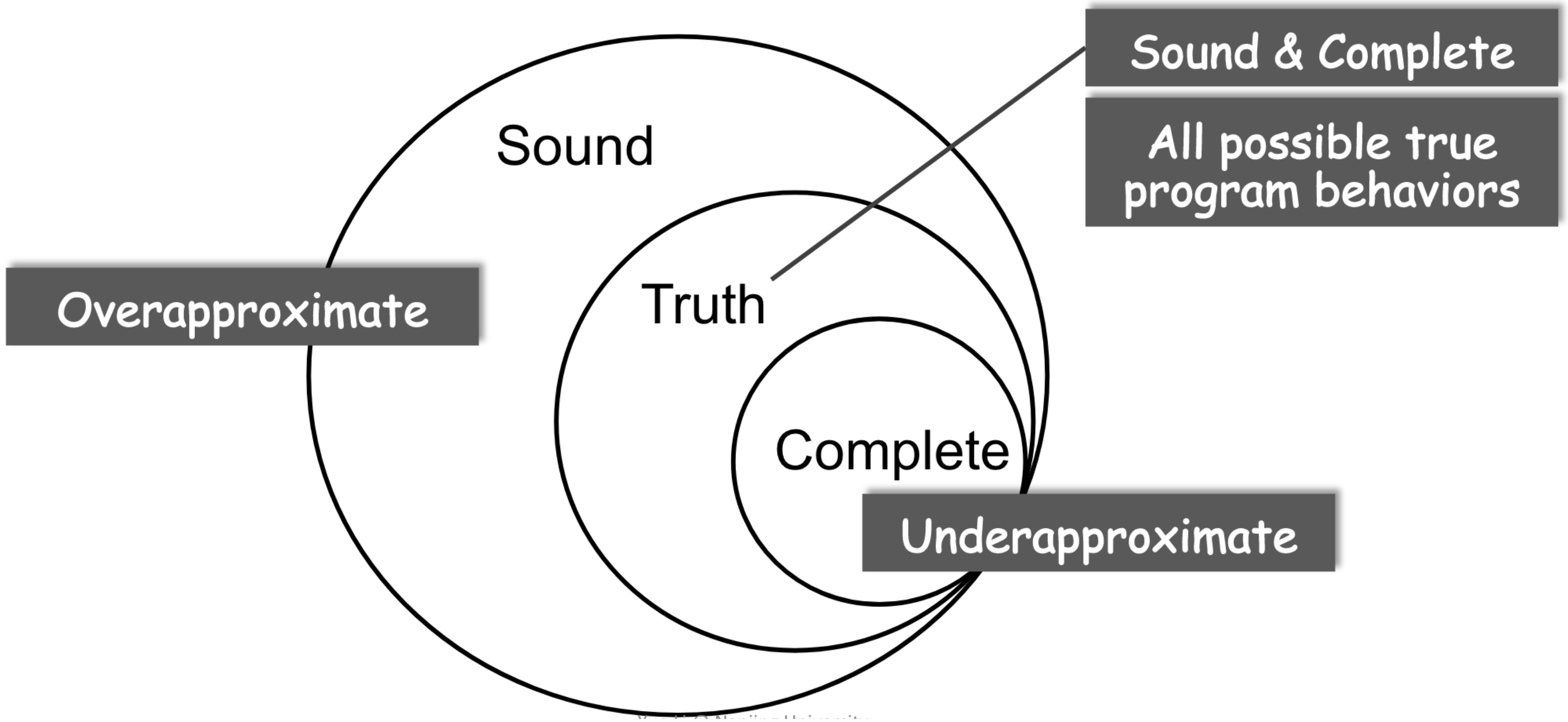
$$F_i \triangleq \underbrace{F_{G_i}}_{\text{neural network}} \wedge \underbrace{\left(\bigwedge_{j=1}^{n} x_{i,j} = v_j^o\right)}_{\text{network input}} \wedge \underbrace{\left(\bigwedge_{j=1}^{m} r_{i,j} = v_{n+j}^o\right)}_{\text{network output}}$$

$$\underbrace{\left(\bigwedge_{i=1}^{n} |x_i - c_i| \leq 0.1\right) \wedge}_{\text{pre-condition}} \underbrace{F_G}_{\text{Neural Network}} \wedge \underbrace{\left(\left(\bigwedge_{i=1}^{n} x_i = v_i^o\right) \wedge (r_1 = v_{n+1}^o)\right)}_{F_1 : r_1 = f(x)} \wedge \underbrace{\left(\left(\bigwedge_{i=1}^{n} c_i = v_i^o\right) \wedge (r_2 = v_{n+1}^o)\right)}_{F_2 : r_2 = f(c)}$$

$$\Rightarrow \underbrace{(\text{class}(r_1) = \text{class}(r_2))}_{\text{post-condition}}$$

Soundness & Completeness

If you are interested, check Rice's Theorem, PL and Static Analysis.



Conclusion

Why Neural Network Verification?

We want to make **NN**

have some desired **properties** we can **formally trust**.

How We Form it?

$pre \wedge \mathbf{F} \Longrightarrow post$

SAT: $\forall n_1, \ldots, nN,$ whether **eval**$(\mathbf{F}(n_1,\ldots,n_N))$=**true**

How We Solve and Use it?

Use SMT to define the **specifications** of our neural network.

Apply SMT solvers to **verify** if our model is robust or of other properties.

Encode nodes, edges, properties and the network.

Questions?

Neural Network Verification IN ACTION

Instructions: https://elio-yang.github.io/res/NNV