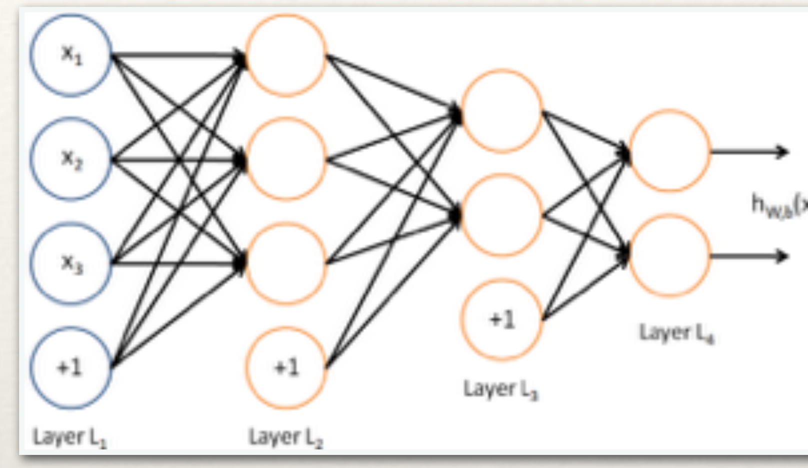
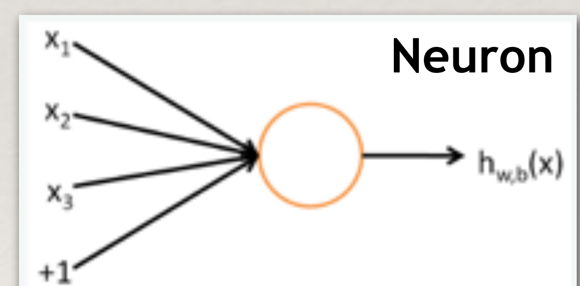
SafeDNN: Understanding and Verifying Neural Networks

Corina Pasareanu (NASA Ames, KBR, CMU)

Artificial Neural Networks

- Computing systems inspired by the biological NNs in animal brains
- Consist of neurons (computational units) organized in **multiple layers**
- Neurons can be active or not; last layer contains decisions
- Perform feature extraction and input transformation
- Learn (progressively improve performance) to do tasks by considering examples
- Can represent complex non-linear relationships





Example activation function: ReLU (Rectified Linear Unit)

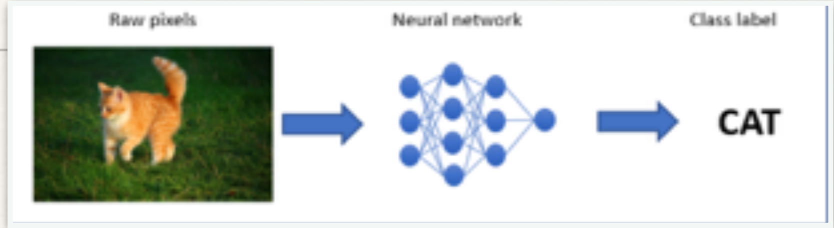
$f(x) = max(0,x)$

$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} Wixi + b)$

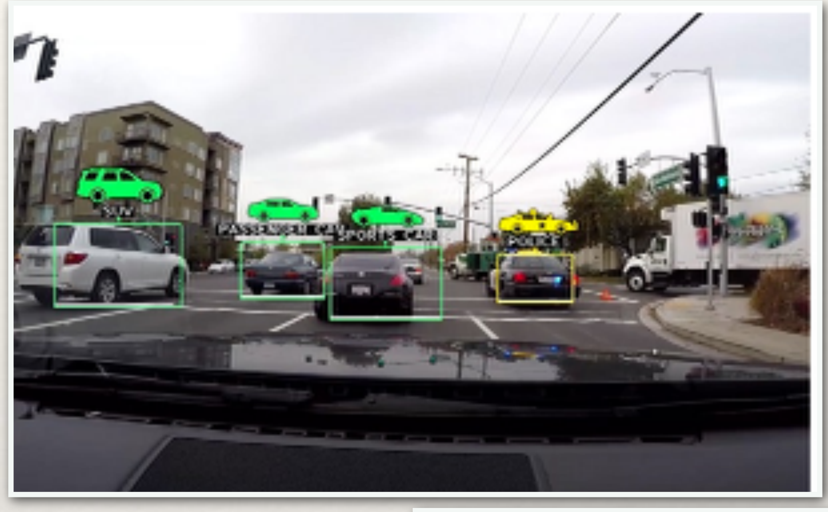
Applications

- Immense popularity …
- Pattern analysis
- Image classification
- Sentiment analysis
- Speech / audio recognition
- Medical diagnosis
- Perception modules in self-driving cars

Image Classification

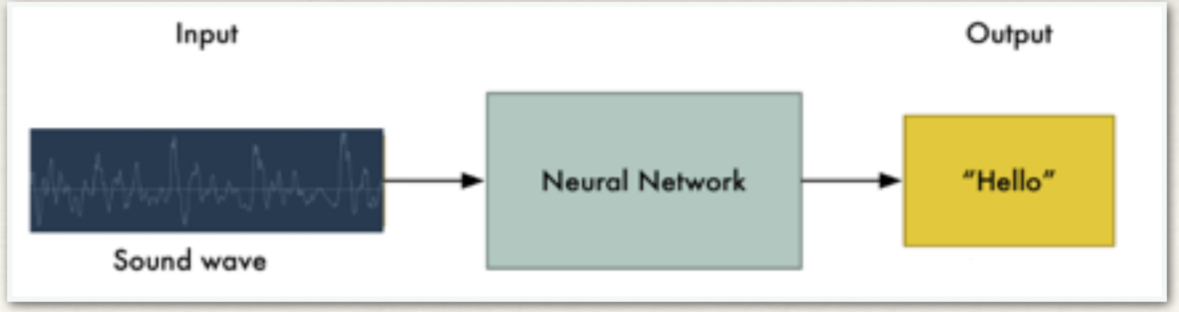


Autonomous Driving



Sentiment Analysis

Speech Recognition



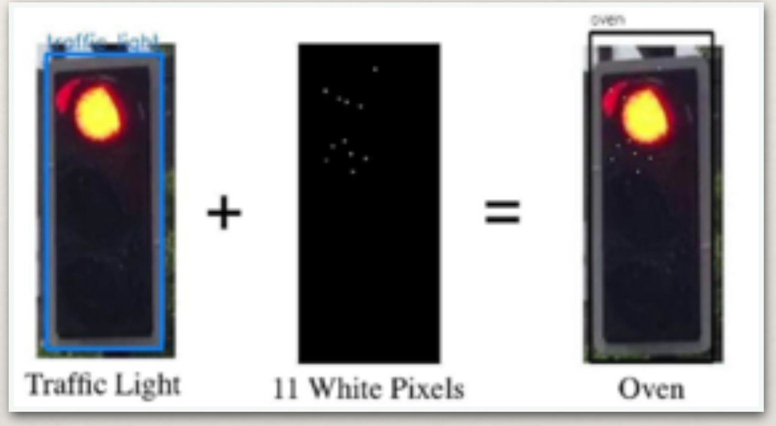
Challenges

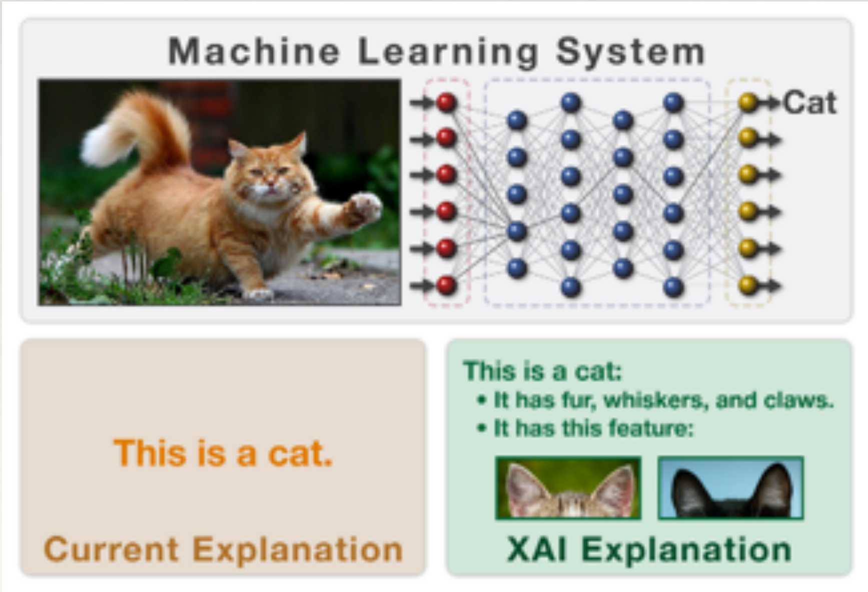
Safety and Security Concerns

- Lack of robustness
  - Small (imperceptible) changes to an input lead to misclassifications
  - Even for highly trained, highly accurate networks
- Lack of explainability
  - It is not well understood why a network gives a particular output
- Lack of formal specifications
  - Networks learn from examples, without high-level specifications
- Scalability
  - Networks are very large, highly interconnected structures; often have huge input spaces; these characteristics prevent thorough verification/testing

What about the data?

- Enough data? Poisoned/unreliable data? Bias?
- Data management?





from DARPA

SafeDNN: Safety of Deep Neural Networks

https://ti.arc.nasa.gov/tech/rse/research/safednn/

- **RSE project**
  - Explores techniques and tools to ensure that systems that use Deep Neural Networks (DNN) are safe, robust and interpretable.
- **Project Members**
  - Corina Pasareanu
  - Divya Gopinath
- **Many students and collaborators**

Recent Advances

Property Inference

- Property Inference for Deep Neural Networks (ASE'19)

Explainability

- A Programmatic and Semantic Approach to Explaining and Debugging Neural Network Based Object Detectors (CVPR'20)

Verification

- Fast Geometric Projections for Local Robustness Certification (ICLR'21)
- NEUROSPF: A tool for the Symbolic Analysis of Neural Networks (ICSE'21, FoMLAS'21)
- DeepCert: Verification of Contextually Relevant Robustness for Neural Network Image Classifiers (SAFECOMP'21)
- Probabilistic Analysis of Neural Networks (SEAMS'20, ISSRE '20)
- Parallelization Techniques for Verifying Neural Networks (FMCAD'20)
- DeepSafe: A Data-Driven Approach for Assessing Robustness of Neural Networks (ATVA'18)

Repair

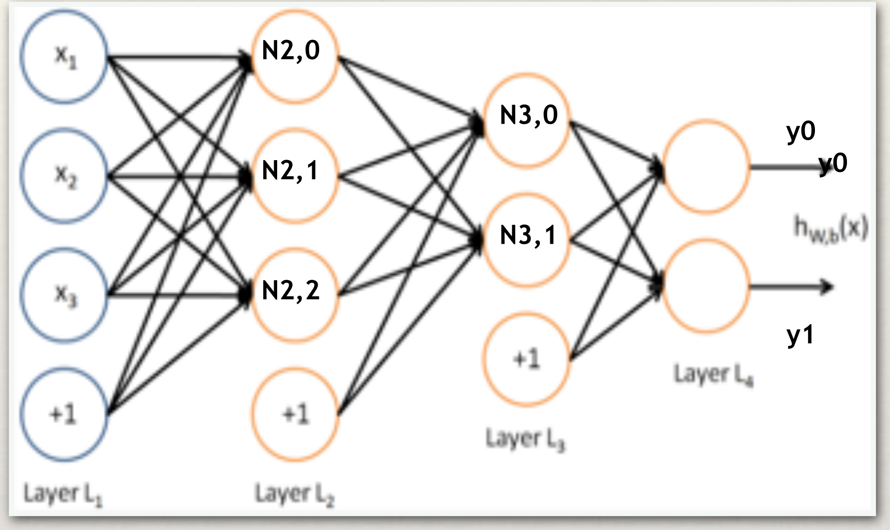
- NNRepair: Constraint-based Repair of Neural Network Classifiers (CAV'21)

Property Inference

Property Inference For Neural Networks

Divya Gopinath, Hayes Converse, Corina S. Pasareanu, Ankur Taly:
Property Inference for Deep Neural Networks. ASE 2019

- **Key Ideas**
  - Infer "likely" properties of a DNN as rules of the form Pre => Post
  - Decomposing a "black-box" model into a set of rules should aid in interpreting and understanding model behavior



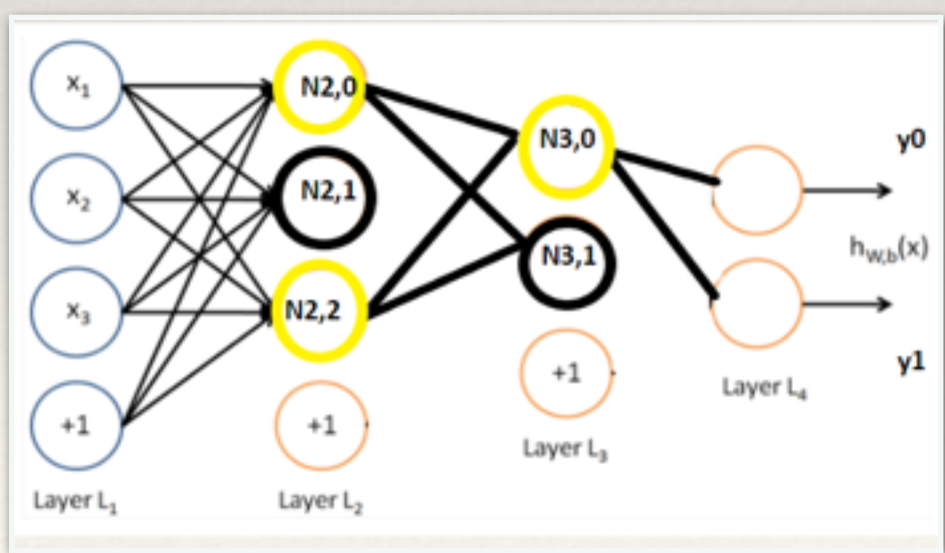
- **Formalizing properties**
  - A constraint in terms of the (on/off) activation patterns of neurons of the network
    - ReLU(x) is on if (x>0) and off if ReLU(x)=0**;** equiv. if (x>0) then x else 0;
    - Piecewise linear nodes equivalent to conditional statements of traditional programs, hence the logic of the network can be captured in the (on/off) activation patterns of neurons
  - Properties can be proved to be valid on the network using a decision procedure (ex. Reluplex), and/or associated with a statistical metric of confidence such as number of satisfying instances

Types of Properties

- **Layer properties** group inputs based on common characteristics at an intermediate layer
  - **Pre** is conjunction of (on/off) constraints on (some/all) neurons of an intermediate layer
  - Intent is to capture properties based on the semantic features the network has learnt
  - Built with decision-tree learning over activations

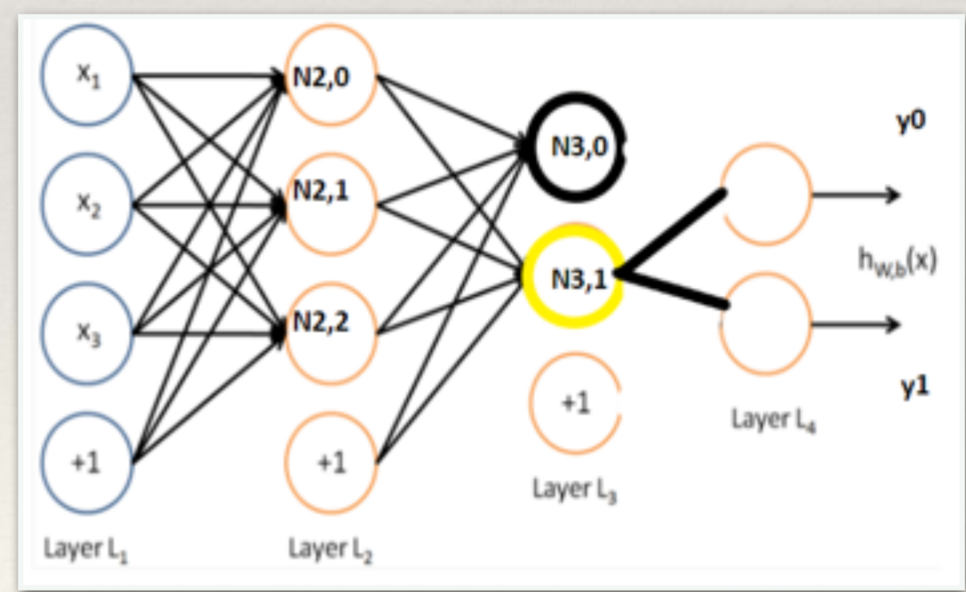
- **Input properties** encode predicates on the input space which imply a certain output property
  - **Pre** is conjunction of constraints on all neurons from the first hidden layer until a certain layer
  - Convex regions of consistent labeling in the input space
  - Built with concolic execution and iterative relaxation

Input Property



(N2,0 > 0 /\ N2,1 = 0 /\ N2,2 > 0 /\ N3,0 > 0 /\ N3,1 = 0) => y0 > y1 (label 0)

Layer Property



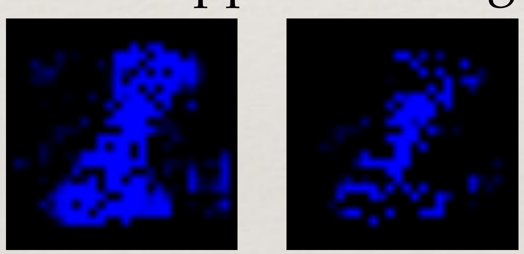
(N3,0 = 0 /\ N3,1 > 0) => y0 < y1 (label 1)

Applications
(Robustness and explanations)

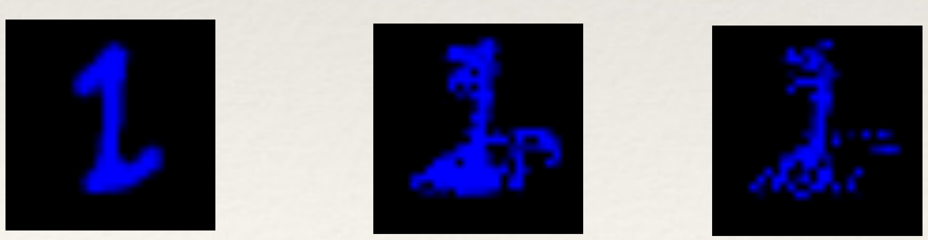
- Provide robustness guarantees
- Generate adversarial examples (cex to Reluplex proofs)

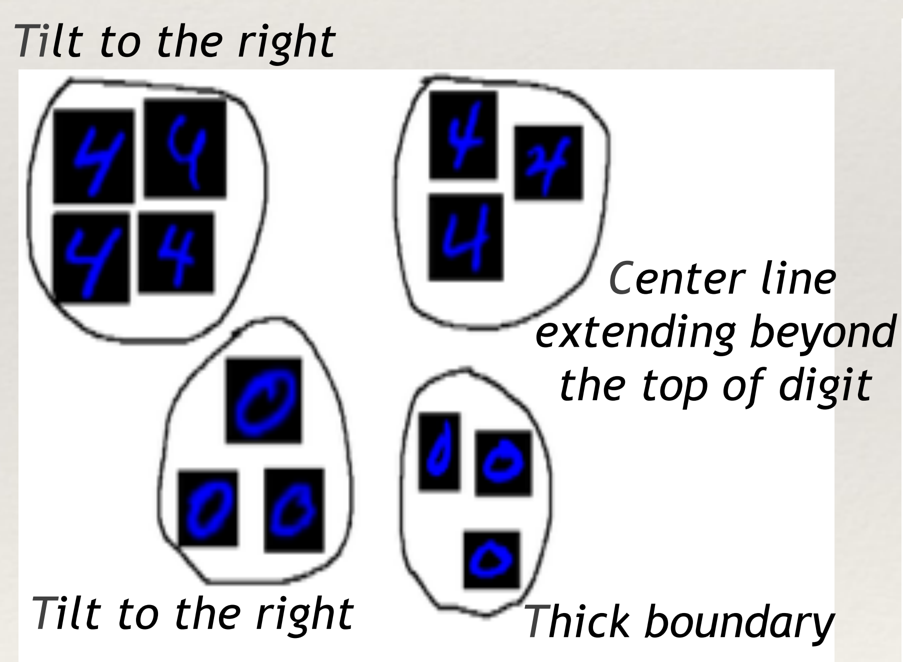
- Formal explanations for perception networks
  - Visualization of multiple images that satisfy the same property and identification of commonality
  - Highlight portions of the image that impact the neurons in the property, akin to attribution techniques
  - Contrast to existing techniques (LIME, Shap) which work on single image

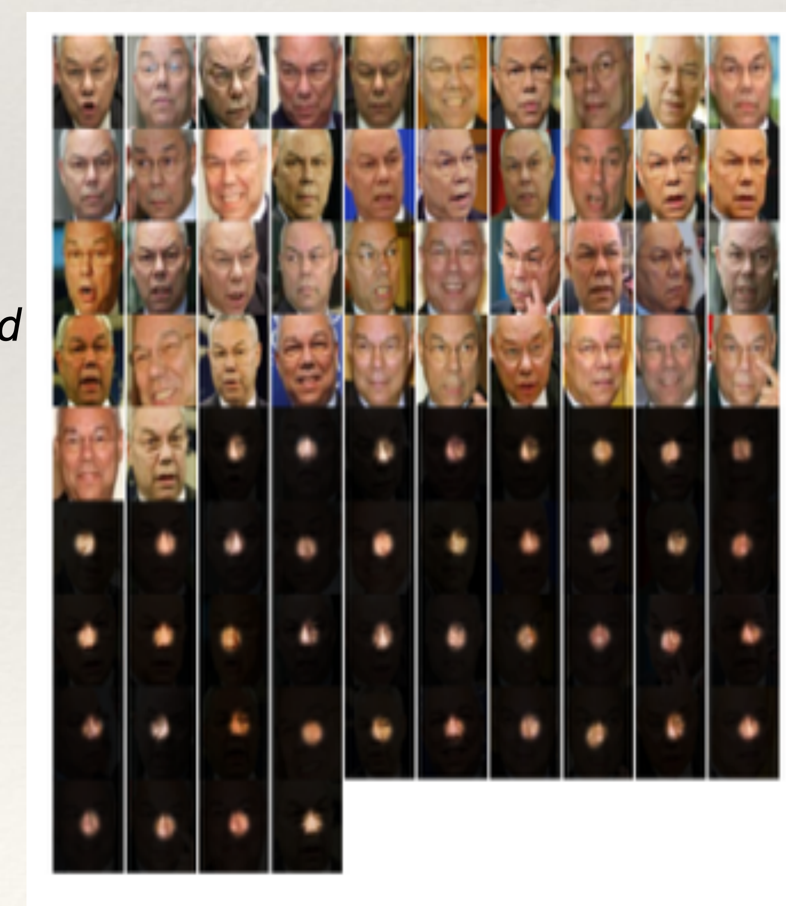
safe under-approximating box



mis-classified input and under-approximating box



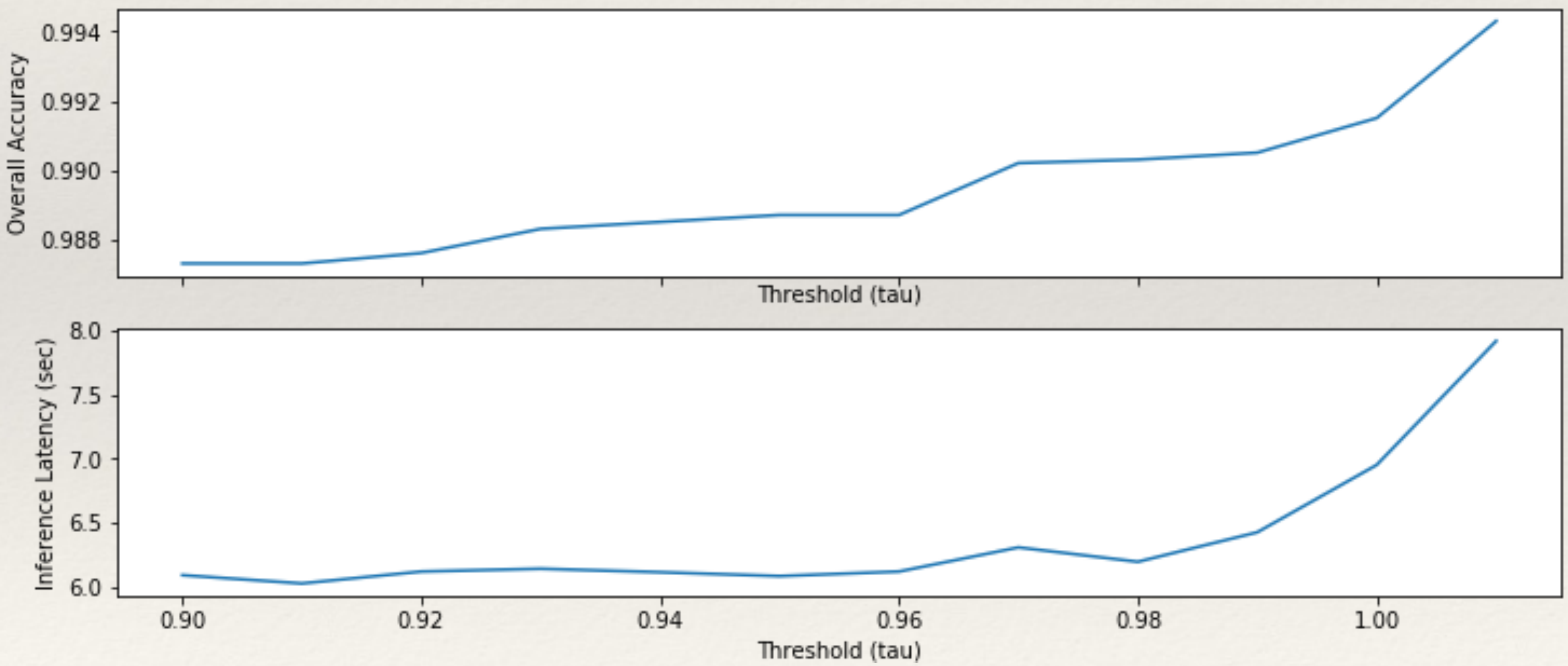




Applications
(Distillation)

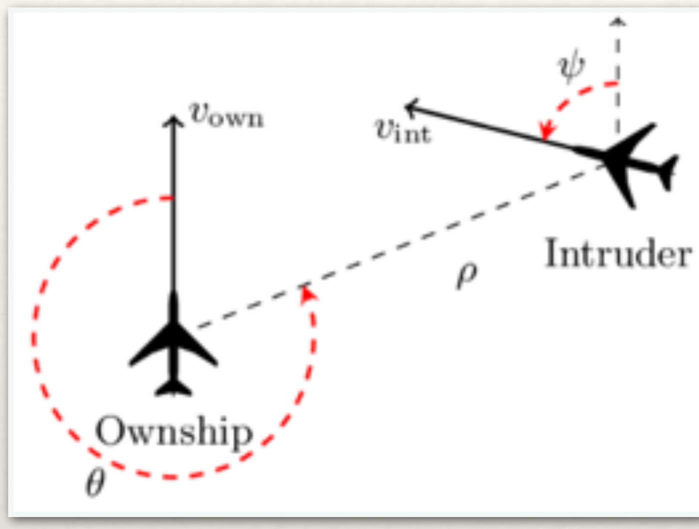
- Build simpler models (distillation)

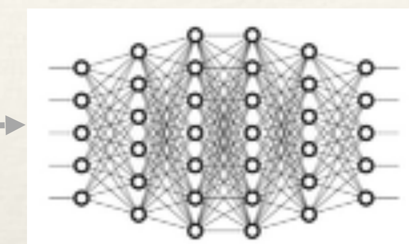
Distillation of an eight layer MNIST network using properties inferred at the first max pooling layer.



Applications
(Property inference, Proof Decomposition)

ACAS-Xu (Airborne Collision Avoidance System-Xu)





DNN controller

- Clear of Conflict (COC)
- Strong Left
- Weak Left
- Strong Right
- Weak Right

- Properties extracted by the approach act as specifications of functionality
  - 31900 ≤ range ≤ 37976, 1.684 ≤ θ ≤ 2.5133, ψ = -2.83, 414.3 ≤ vown ≤ 506.86, vint = 300, has turning advisory **COC**
  - range = 499, -0.314 ≤ θ ≤ -3.14, -3.14 ≤ ψ ≤ 0, 100 ≤ vown ≤ 571, 0 ≤ vint ≤ 150, has turning advisory **Strong Left**
  - range = 48608, θ = -3.14, ψ = -2.83, vown(full range), vint (full range) has turning advisory **COC**

- Decomposed proofs of properties of the form A => B , using "layer patterns" σ,
  - by checking A => σ and σ => B separately w/ Reluplex;
  - significant **speedup** obtained; checked property that timed out with monolithic verification

Explainability

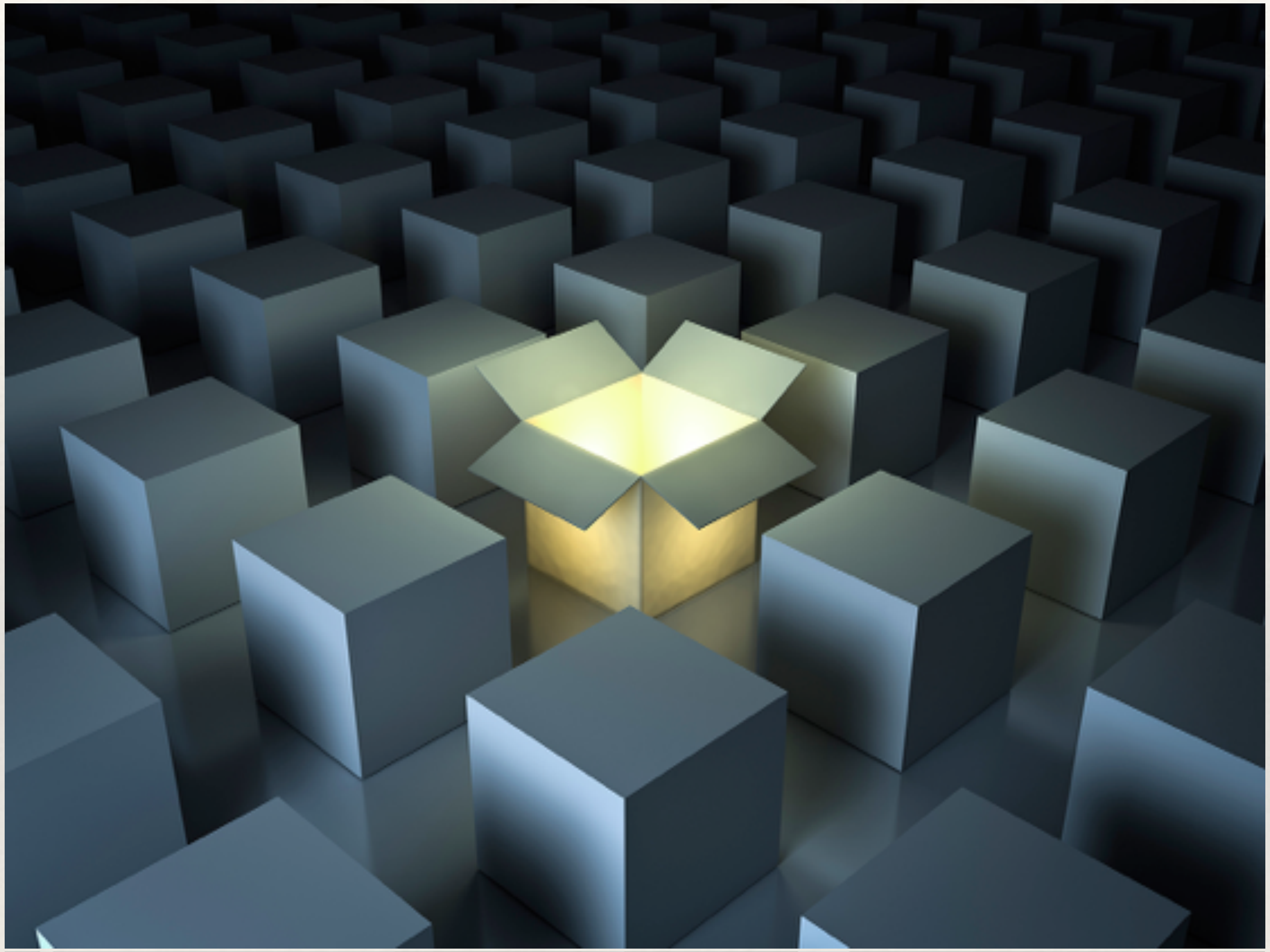
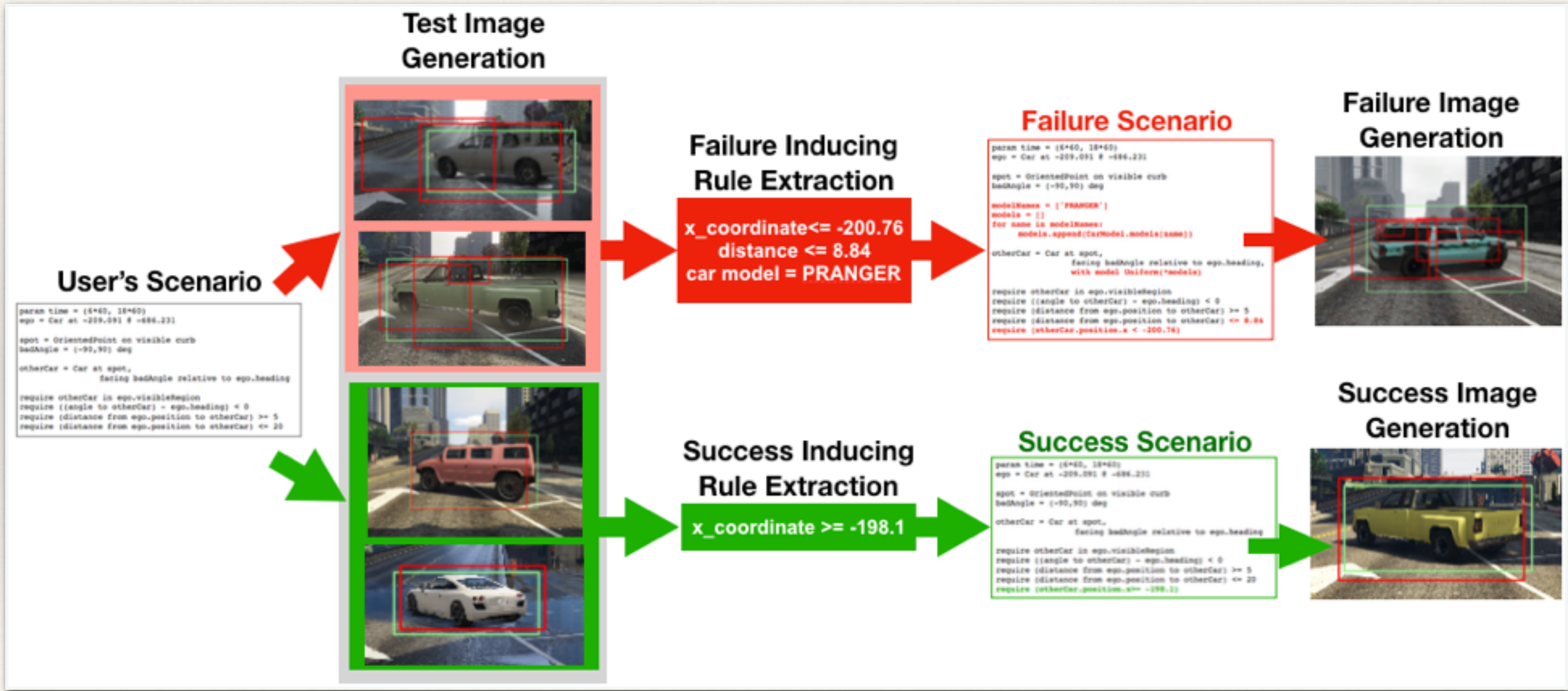


Image: iStock Phot

Extracting Semantic Explanations of a Detection Module

Edward Kim, Divya Gopinath, Corina S. Pasareanu, Sanjit A. Seshia:
A Programmatic and Semantic Approach to Explaining and Debugging Neural Network Based Object Detectors. CVPR 2020



Key idea: leverage high-level semantic features encoded in a SCENIC program to derive rules (sufficient conditions) that explain the module; rules generated with decision tree learning, anchors and activation patterns
Benefits: better explain and debug the module.

Results

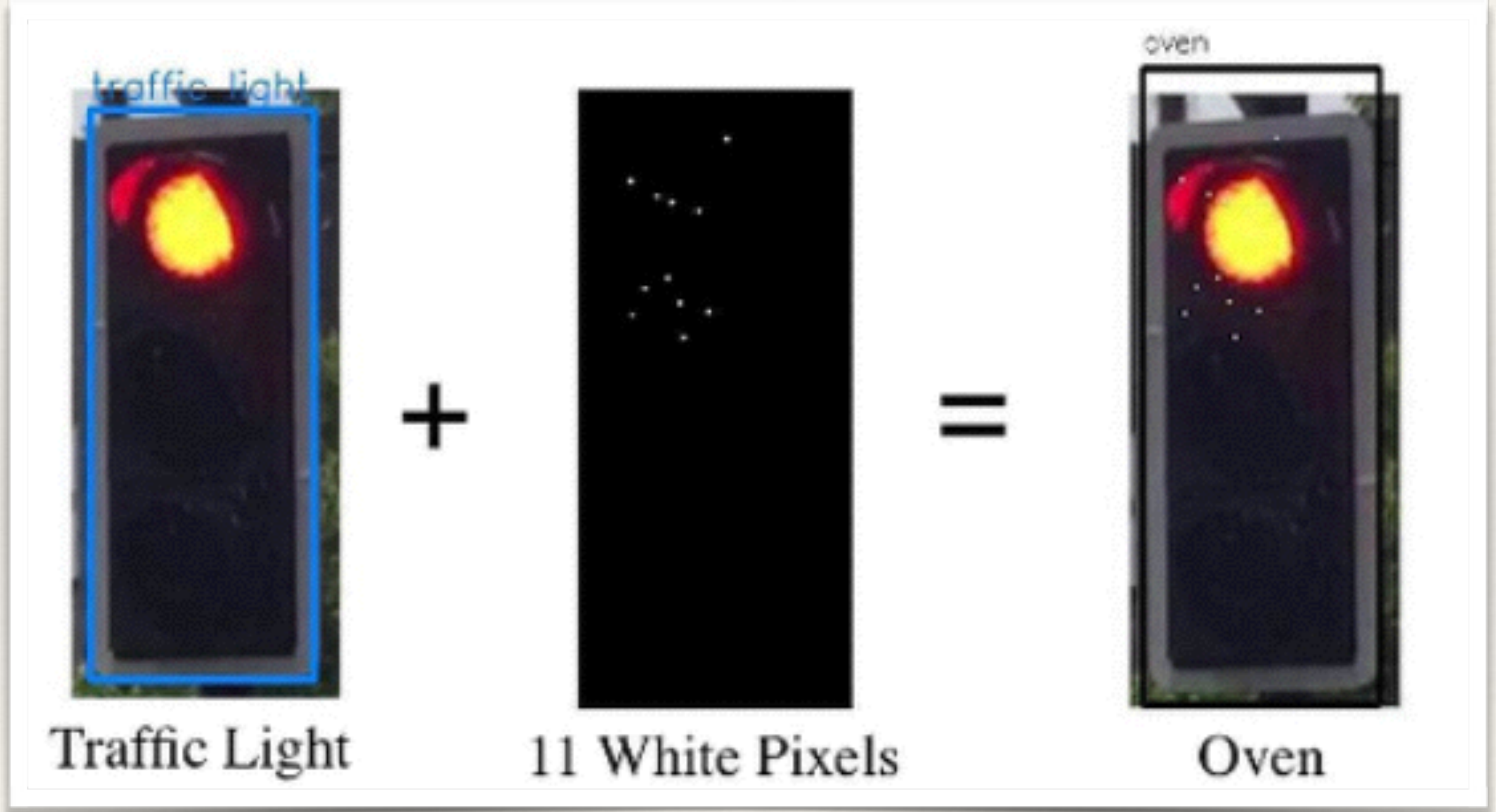
Rules for correct detection

Scenario # (Baseline→Rule Precision)	Rules
Scenario 1 (65.3% → 89.4%)	**x coordinate** $\geqslant$ -198.1
Scenario 2 (72.3% → 82.3%)	**hour** $\geqslant$ 7.5 $\wedge$ **weather** = all except neutral $\wedge$ **car0 distance from ego** $\geqslant$ 11.3m $\wedge$ **car0 model** = {Asea, Bison, Blista, Buffalo, Dominator, Jackal, Ninef, Oracle}
Scenario 3 (61.7% → 79.4%)	**car0 red color** $\geqslant$ 74.5 $\wedge$ **car0 heading** $\geqslant$ 220.3 deg
Scenario 4 (89.6% → 96.2%)	**car0 model** = {Asea, Baller, Blista, Buffal, Dominator, Jackal, Ninef, Oracle}

Rules for incorrect detection

Scenario # (Baseline→Rule Precision)	Rules
Scenario1 (34.7% → 87.2%)	**x coordinate** $\leqslant$ -200.76 $\wedge$ **distance** $\leqslant$ 8.84 $\wedge$ **car model** = PRANGER
Scenario 2 (27.7% → 44.9%)	**hour** $\geqslant$ 7.5 $\wedge$ **weather** = all except Neutral $\wedge$ **car0 distance from ego** < 11.3
Scenario 3 (38.3% → 83.4%)	**weather** = neutral $\wedge$ **agent0 heading =** $\leqslant$ 218.08 deg $\wedge$ **hour** $\leqslant$ 8.00 $\wedge$ **car2 red color** $\leqslant$ 95.00
Scenario 4 (10.4% → 57.3%)	**car0 model** = PATRIOT $\wedge$ **car1 model** = NINEF $\wedge$ **car2 model** = BALLER $\wedge$ 92.25 < **car0 green color** $\leqslant$ 158 $\wedge$ **car0 blue color** $\leqslant$ 84.25 $\wedge$ 178.00 < **car2 red color** $\leqslant$ 224

Verification

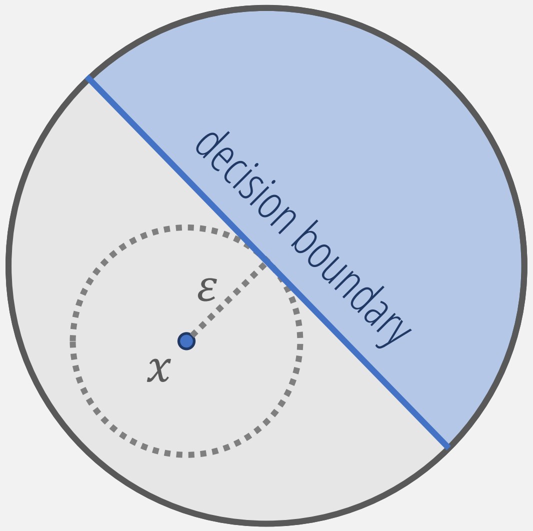


Fast Geometric Projections for Local Robustness Certification

Aymeric Fromherz, Klas Leino, Matt Fredrikson, Bryan Parno, Corina S. Pasareanu: Fast Geometric Projections for Local Robustness Certification. ICLR 2021

- A model F satisfies *local robustness* with robustness radius ε on a point x if

$$\forall x'. \|x - x'\|_p \leq \varepsilon \implies F(x) = F(x')$$



- Valid for any norm, but we focus on the ℓ_2 norm, which is less well-studied

Defenses

Heuristic

- Adversarial training
- TRADES

Certification

training procedure

- Kolter-Wong
- Maxim Margin Regularization

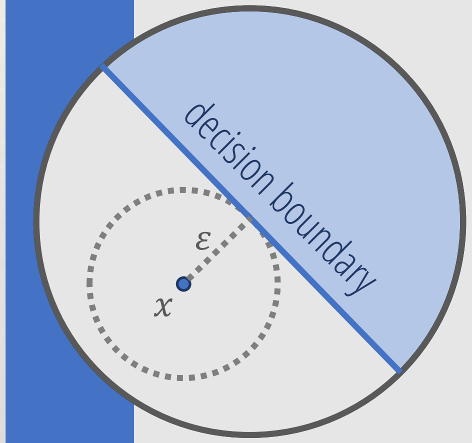
model-agnostic verification

- GeoCert
- MIP
- …

Probabilistic

- Randomized Smoothing

Certification of Local Robustness



$$\forall x'.\ \|x - x'\|_p \le \varepsilon \implies F(x) = F(x')$$

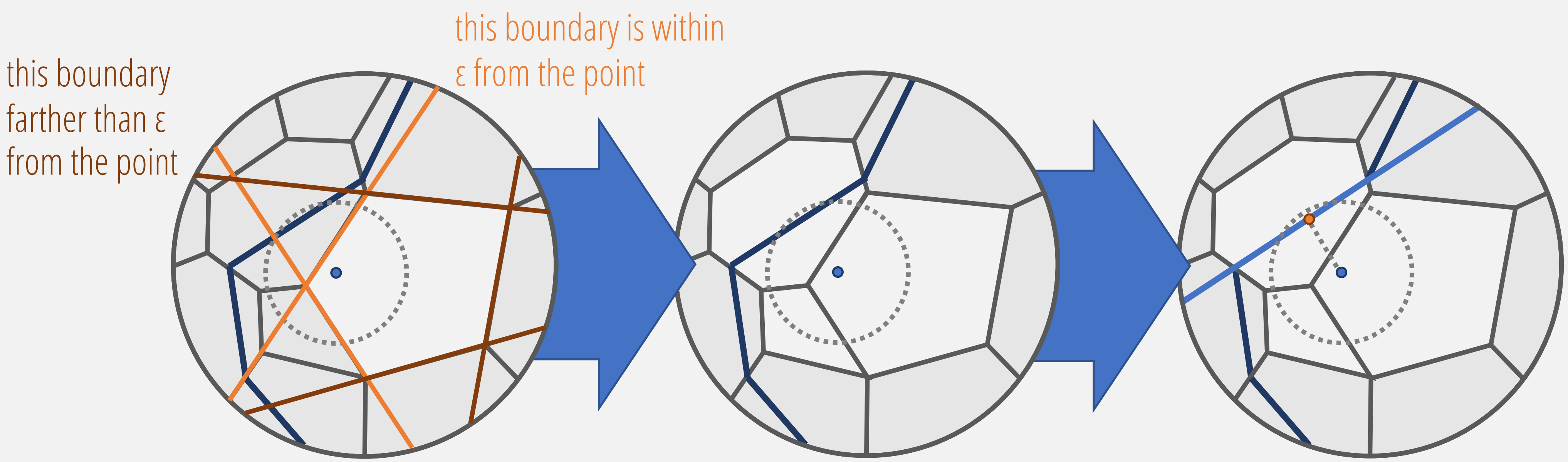
Idea: use a more refined understanding of the *geometry* of a class of networks

ReLU Networks as Polyhedral Complex

- Piecewise linear networks partition input domain into a polyhedral complex
  - Input regions correspond to activation patterns
  - Boundaries of regions can be computed with gradients
- Given a region, can compute distance to boundary using constraint solving (e.g., GeoCert, MIP): **expensive**
- **Our contribution:**
  - Use geometric projections (no constraint solving)
  - Acceleration with GPUs
  - Sound but not complete

Fast Geometric Projections (FGP) Method

Projections offer a fast, sound way to see which boundaries are within our ε-radius



begin by *exploring* the starting region: for each boundary of starting region, check if the boundary is in the ε-ball

explore each of the neighboring regions whose boundaries were in the ε-ball

if a decision boundary is found, project onto it to verify an adversarial example was found

Results

- On adversarially-trained dense networks, FGP outperforms GeoCert by **3 orders of magnitude** and MIP by **4 orders of magnitude**
- UNKNOWN results account for **only 3-5% of cases**, while GeoCert and MIP time out on 10-100% of cases

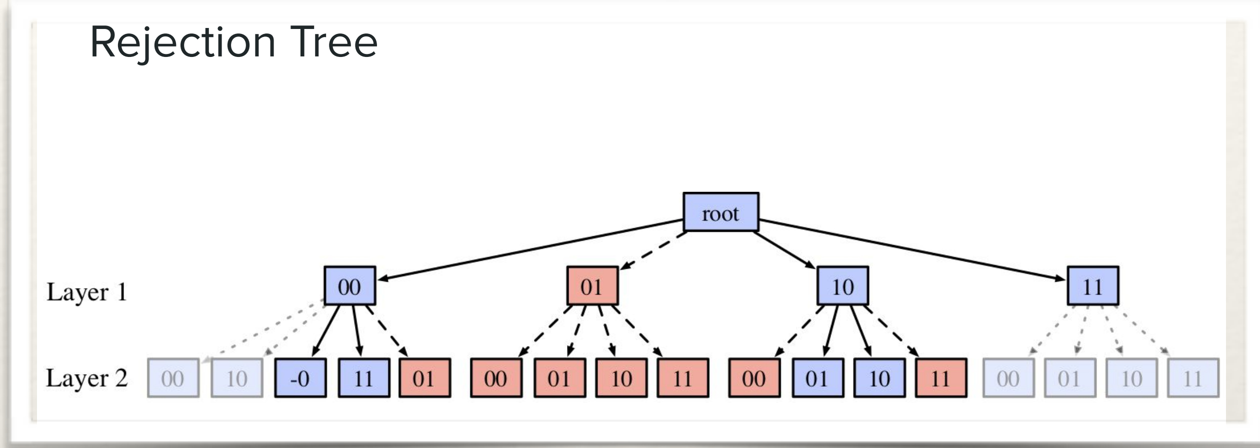
Probabilistic Analysis of Neural Networks

Hayes Converse, Antonio Filieri, Divya Gopinath, Corina S. Pasareanu: Probabilistic Symbolic Analysis of Neural Networks. ISSRE 2020

- *Properties of Neural Networks*
  - Proved with formal verification tools (Reluplex/Marabou from Stanford)
  - Properties often do not hold; point-wise robustness checks output binary answers but lack detail; verification tools do not scale
- *Probabilistic properties*
  - More natural, e.g. accuracy
  - Checked with statistical methods: scale but provide no guarantees, tend to ignore “rare” events
- *Our proposition*
  - Probabilistic analysis through *symbolic execution* and *volume computations*
  - **Benefits**: increase impact of sampling and provide precise confidence
  - Collect mathematical constraints along neuron activations and apply volume computations to compute probabilities

Technique

- Symbolically / concolically execute concrete inputs
- Observe activation patterns; organize them in a tree
- *Reject* inputs that add no information (i.e., previously seen activation patterns)



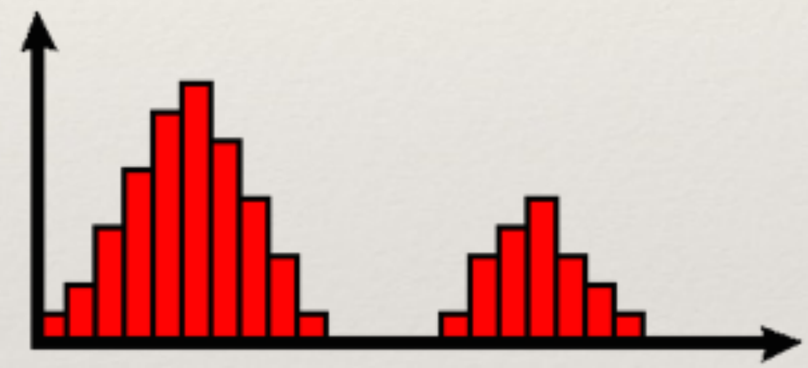
- Add decision conditions to constraints based on network output (logits) layer
- Compute volume of constraints
- Stop at user defined criterion (coverage, number of paths, rejection percentage, …)

- Similar to previous work on probabilistic symbolic execution, but adapted to neural networks

Input Distributions and Probabilities

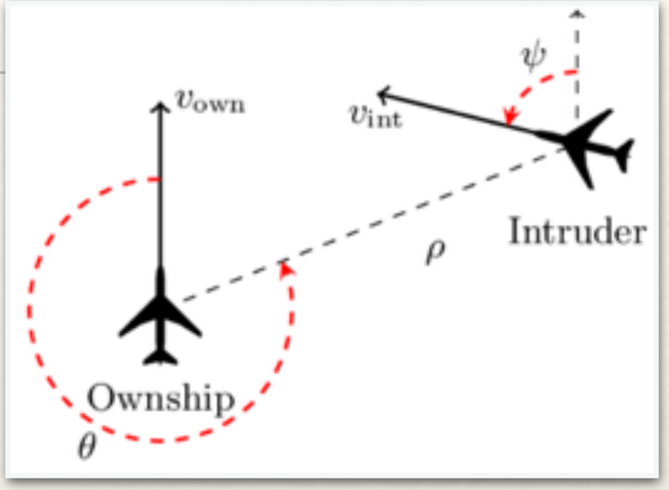
- Uniform distribution:
  - *Pr(D)=Vol(constraints for D)/Vol(full domain)*
- Non-uniform distribution: partition input domain, create histogram distribution: (s_i, p_i)

$$Pr(\mathcal{D}) = \sum_{s_i} p_i \cdot \sum_{AC \rightsquigarrow \mathcal{D}} \frac{Vol(AC \wedge s_i)}{Vol(\mathbb{D}_x)}$$



  - *AC* are activation conditions (together with decision conditions) leading to event *D*
  - Confidence:
    - % of input domain covered by the analysis

Applications



- Implemented techniques in *SpaceScanner*
- Robustness/sensitivity analysis for ACAS-Xu
  - DNN controllers in next-generation Airborne Collision Avoidance Systems for unmanned aircraft
- Fairness analysis for decision making networks
- **Results for ACAS-Xu**
  - Found the network to be highly robust in assigning Clear-of-Conflict (COC) decisions
  - Found the network to be more **vulnerable** to adversarial perturbations for the advisories weak-left, strong-left and strong-right
  - Statistical analysis produces comparable results but **misses cases** when probability of misclassification is non-zero

Repair

NNRepair: Constraint-based Repair of Neural Network Classifiers

Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, Corina S. Pasareanu: NNrepair: Constraint-based Repair of Neural Network Classifiers, CAV'21

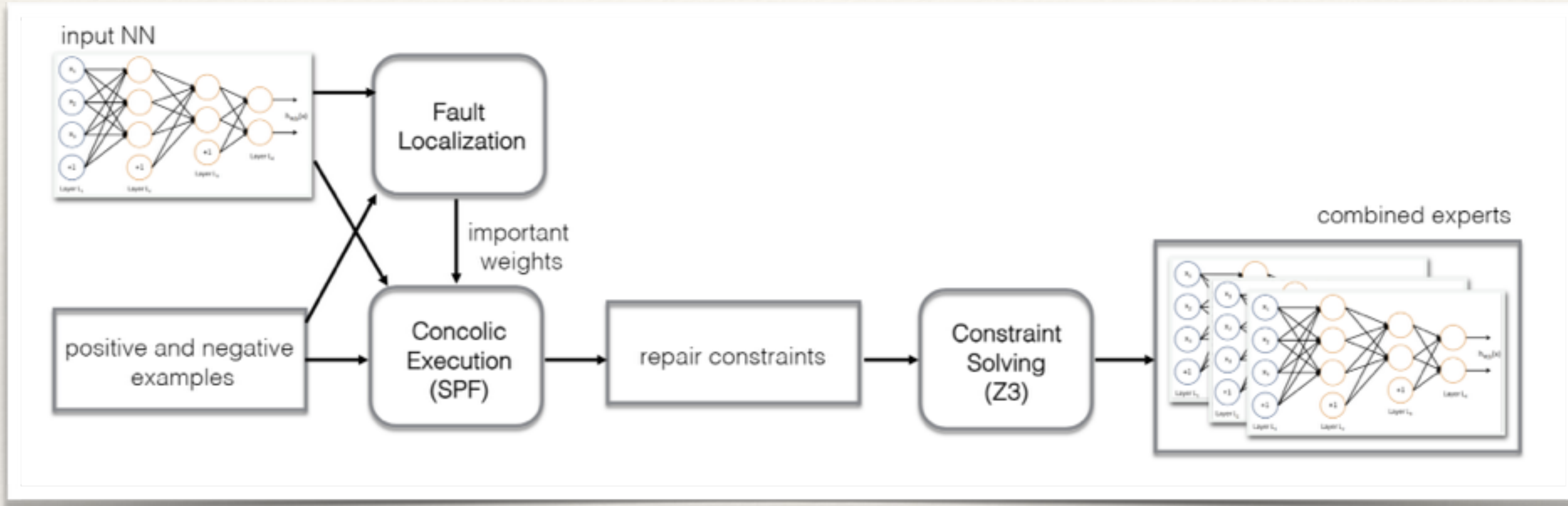
- Problem: The network is faulty
  - Low accuracy, lack of robustness, poisoned training data
- Retraining could be used to alter the neural network parameters and repair for faults.
  - Difficult and expensive subject to uncertainties.
  - Result in a network that is quite different from the original one.
  - May not be possible (in the absence of additional data)
- NNrepair: constraint solving for repairing neural networks
- Similar to traditional program repair.
  - **Fault localization** identifies the network parameters that are the likely source of defects.
  - **Repair** uses constraint solving to apply small modifications to the network **parameters** to remedy the defects.

Types of Repair

- **Last-layer repair**
  - Attempts to modify the decision constraints at the last layer.
  - For last-layer repair, the oracle of the repair is the desired label.

- **Intermediate-layer repair:**
  - Attempts to fix failures by modifying the behavior of neurons at an inner layer of the network.
  - For intermediate-layer repair, the oracle for the repair is the "activation pattern"; keeps the repair local
  - Potentially more scalable

Framework



Repair constraints encode network decision for **positive examples** and modify (i.e., correct) network decision for **negative examples**

Example: Intermediate-layer Repair

- Consider input $X_4 = [1.5; 2.0]$
- It is misclassified to “1” (ideal is “0”)
- For all the inputs correctly classified to “0”, the neuron pair **(N_2, N_3)** in second layer has activation pattern **(off, on)**
- For the failing input, this pattern is not satisfied; the activation for (N_2, N_3) is (on, on)

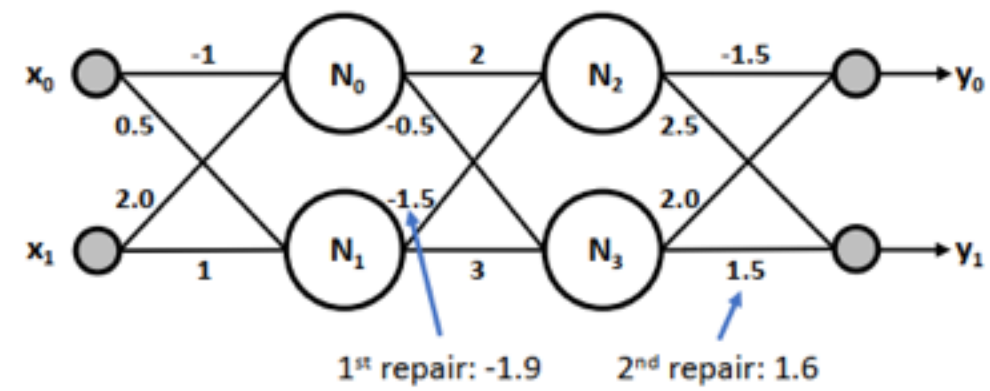


Fig. 1. Example

Table 1. Data for Example

	x_0	x_1	N_0	N_1	N_2	N_3	y_0	y_1	class	ideal
x_0	1	1	1	1	0	1	8	6	0	0
x_1	0	1	1	1	1	1	0.25	9.25	1	1
x_2	1	0	0	1	0	1	3	2.25	0	0
x_3	-1	1	1	1	1	0	-7.87	13.12	1	1
x_4	1.5	2	1	1	1	1	12.68	12.68	1	0
after repair:	1.5	2	1	1	0	1	13.3	10.5	0	0
x_5	0.6	1	1	1	1	1	5.91	5.62	0	1
after repair:	0.6	1	1	1	1	1	5.91	5.95	1	1

Example: Intermediate-layer Repair

- Modify the neuron activations of the second layer on the failing input to satisfy pattern (off,on)
  - Identify the weights to be modified using an attribution-based approach
  - Use constraint solving to compute the values of the new weights
- Changing the weight of the edge connecting N_1 and N_2 from -1.5 to -1.9 changes the activation pattern for (N_2; N_3) to (off, on) on the failing input
- Preserves the behavior of the neurons (their activation pattern) and the output of the model on passing inputs

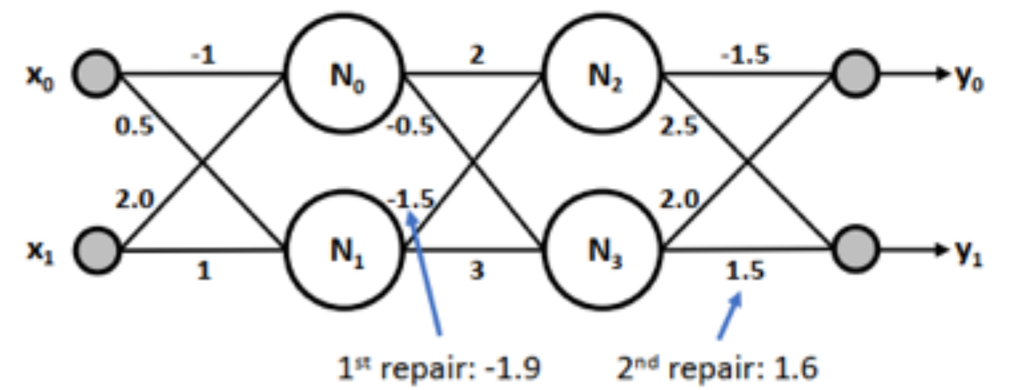


Fig. 1. Example

Table 1. Data for Example

	x_0	x_1	N_0	N_1	N_2	N_3	y_0	y_1	class	ideal
x_0	1	1	1	1	0	1	8	6	0	0
x_1	0	1	1	1	1	1	0.25	9.25	1	1
x_2	1	0	0	1	0	1	3	2.25	0	0
x_3	-1	1	1	1	1	0	-7.87	13.12	1	1
x_4	1.5	2	1	1	1	1	12.68	12.68	1	0
after repair:	1.5	2	1	1	0	1	13.3	10.5	0	0
x_5	0.6	1	1	1	1	1	5.91	5.62	0	1
after repair:	0.6	1	1	1	1	1	5.91	5.95	1	1

Results

- Demonstrated our technique in the context of three different scenarios:
    - Improving the overall accuracy of a model
    - Fixing security vulnerabilities caused by poisoning of training data

    - Improving the robustness of the network against adversarial attacks

- **NNrepair** can improve the performance of the network by **45.56%** on poisoned data and **10.40%** on adversarial data.

Other Repair Techniques

- MODE [Ma et al. ESEC/FSE'18]: differential analysis + retraining
  - NNRepair has similar performance: better on MNIST-HQ but worse on MNIST-LQ
- Apricot [Zhang et al. ASE'19] generates a set of reduced models and repairs weights based on average weight of reduced models
- Sotoudeh and Thakur [2019] uses SMT solving to repair ACASXu networks
- Other …

- None of these techniques address all three scenarios that we consider
- Previous techniques focus only on last layer repair

Future Work

Future Work

- Automated repair for poisoned NN
- Structural testing coverage for neural networks
- Learning with formal guarantees
- Relating NN properties to system-level properties of an autonomous system

Thank you!

https://ti.arc.nasa.gov/tech/rse/research/safednn/