

CONSTRAINT-BASED REPAIR OF NEURAL NETWORK CLASSIFIERS

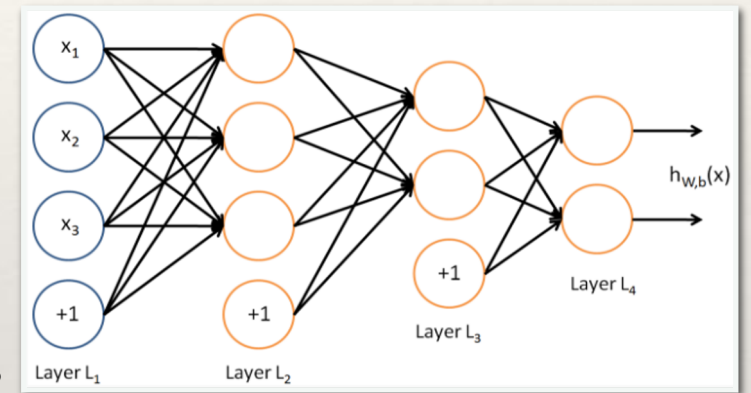
Divya Gopinath

Research Scientist, RSE group NASA AMES, KBR Inc.

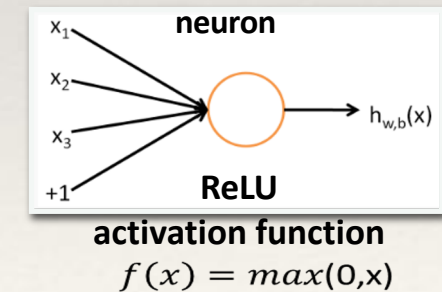
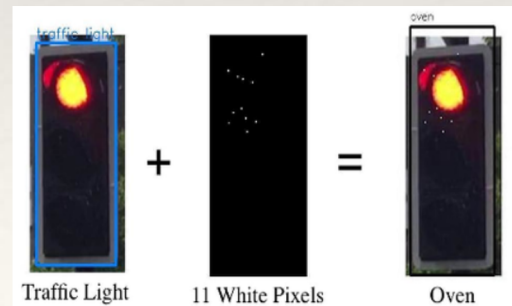
Deep Neural Networks

- **Deep Neural Networks (DNNs)** have gained immense popularity in recent years
 - Image classification, sentiment analysis, autonomous driving so on
- Despite widespread usage, there are safety and security concerns
 - *Lack of guarantees for network behavior*; no formal specifications of correctness
 - *Vulnerable to adversarial perturbations, backdoor attacks*
 - *Design not amenable to reasoning*; data-driven logic that is not interpretable, very huge state spaces

Deep Neural Network



Adversarial Attacks



Symbolic Path Finder for DNN analysis

- Techniques that enable analysis of DNNs for purposes such as debugging, verification, repair are the need of the hour
- The wealth of techniques for traditional software program analysis can be leveraged
- **Symbolic Path Finder (SPF)** builds on the JPF framework to enable symbolic execution, along with model checking in Java byte-code programs
 - Automated test case generation, Error detection so on
- Utilizing the SPF framework for the analysis of DNN models would be very beneficial
 - *NeuroSPF** is a tool that converts Keras models to Java programs thereby enabling the application of SPF to analyze the behavior of DNN models
- **NNRepair: Constraint-based Repair of Neural Network Classifiers***, explores the use of SPF to enable debugging and repair of faulty neural network models

* Muhammad Usman, Yannic Noller, Corina S. Pasareanu, Youcheng Sun, Divya Gopinath: NEUROSPF: A Tool for the Symbolic Analysis of Neural Networks. ICSE (Companion Volume) 2021

* Muhammad Usman, Divya Gopinath, Youcheng Sun, Yannic Noller, Corina S. Pasareanu: NNrepair: Constraint-Based Repair of Neural Network Classifiers. CAV 2021

Debugging and Repair of DNN models

- **Problem**

- The network is **faulty**; **low overall accuracy**, **lack of robustness to specific types of inputs**; **adversarial**, **poison**
- Faulty network components; architecture, learnable parameters (weights, biases)
- Given a set of failing tests and passing tests, modify network components to improve accuracy

- **Existing techniques**

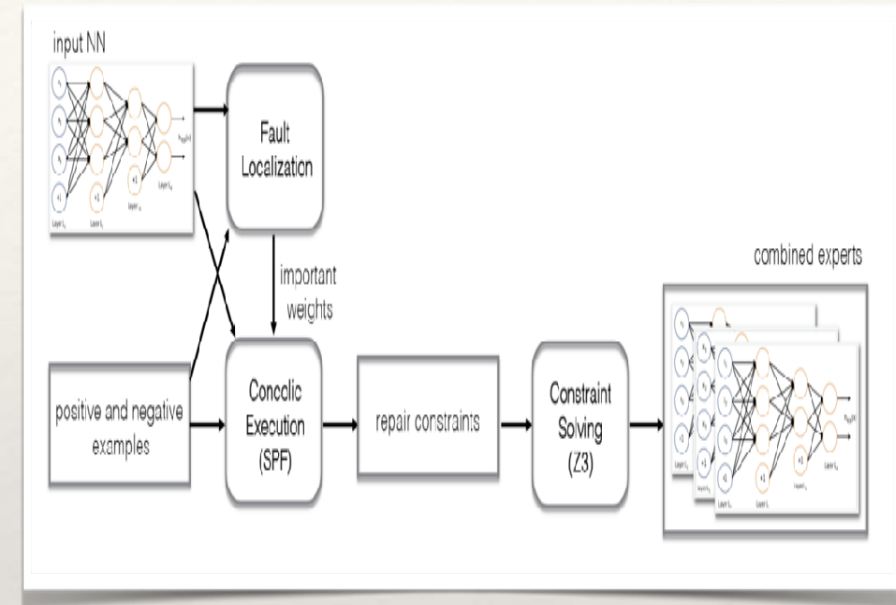
- Retraining is a commonly adopted technique but is expensive, subject to uncertainties, may require additional data
- MODE [Ma et al. ESEC/FSE'18] adopts differential-analysis to identify buggy neurons and performs guided retraining
- Apricot [Zhang et al. ASE'19] generates a set of reduced models and repairs weights based on average weight of reduced models
- Recent work Sotoudeh and Thakur [2019] uses SMT solving to repair networks to satisfy correctness specifications; mainly shown to work on ACASX network (low input dimensionality)

- **Our Approach: NNRepair**

- Precise fixing of errors with no additional data
- Minimal changes to the network
- Enables repair at the last layer and intermediate layer
- Use of inferred properties as oracles for repair

NNRepair Framework

- **Inputs:** DNN classifier model, positive and negative examples
- **Steps:**
 - DNN model *translated* into a Java Program
 - **Fault Localization:**
 - Modify *weights* of the network
 - Restrict the repair to a single layer; *one of the dense layers, or the last layer*
 - *Prophecy** used to *extract patterns* for correct classification to every class at the intermediate layers
 - Identify *suspicious neurons* and *incoming suspicious edges*
 - **Concolic Execution using SPF:**
 - Symbolic deltas added to suspicious weights
 - Program executed on positive and negative examples to collect repair constraints
 - **Constraint Solving using Z3:**
 - Find solutions for deltas that correct behavior on the negative examples while retaining behavior on positive examples

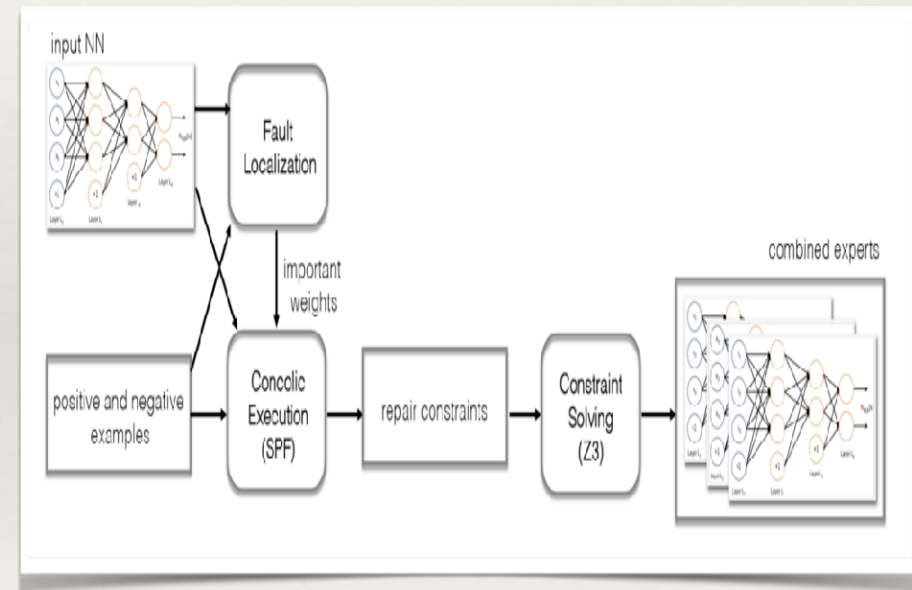


Example Patterns

$$(N_{1,3} = 0 \wedge N_{1,1} \geq 0 \wedge N_{1,2,1} = 0) \Rightarrow \text{label } 0$$
$$(N_{1,3} \geq 0 \wedge N_{1,10} = 0 \wedge N_{1,0} \geq 0) \Rightarrow \text{label } 1$$

NNRepair Framework

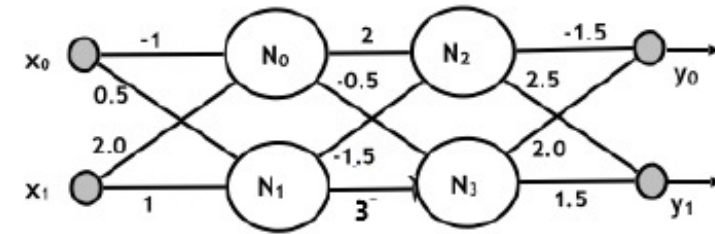
- **Steps:**
 - **Generate expert networks:**
 - Bottleneck is the solving time of repair constraints
 - May not be able to find single value for each suspicious weight that improves accuracy for all labels
 - Solve the repair constraints for each label separately generating expert networks
 - **Combine expert networks:**
 - *Execute all experts at inference time* in a combined manner
 - No expert network predicts its label for the input → Roll-back to original model output
 - Expert Network L predicts label L for the input → Output label is L
 - More than 1 expert networks predict their label for the input → Resolve Conflict using either *Naïve*, *Confidence*, *Voting* strategies
 - *Merge all experts (before inference time)* by setting the weight to average of all experts
 - *Filter out weak experts* (based on F1 over positive and negative examples for all labels) and then follow either of the above two



Example: Intermediate-layer Repair

- Consider input $X_4 = [1.5; 2.0]$, misclassified to “1” (ideal is “0”)
- Pattern for correct behavior:** For all the inputs correctly classified to “0”, the neuron pair (N_2, N_3) in the second layer has activation pattern (*off, on*)
- Localization:**
 - For the failing input, the correct behavior pattern is not satisfied; the activation for (N_2, N_3) is (*on, on*), identifying **N2** as the mis-behaving neuron
 - Simple statistical method to identify *the important weights* which impact the N2’s output more for the failing tests than passing tests

$$Score(E_i) = \frac{\sum_{X \in Fail} |N_i(X) \cdot w_i|}{\#Fail} - \frac{\sum_{X \in Pass} |N_i(X) \cdot w_i|}{\#Pass}$$



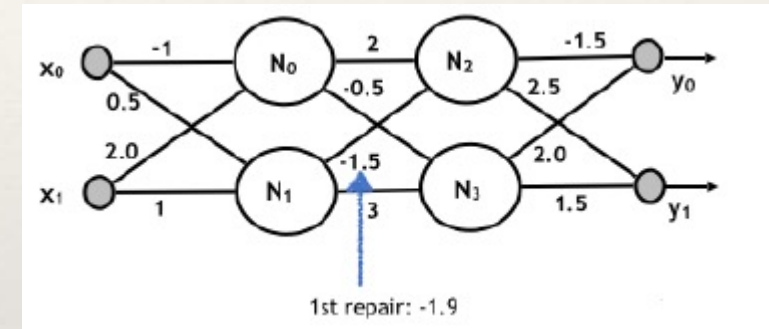
	x0	x1	N0	N1	N2	N3	y0	y1	class	ideal
x0	1	1	1	1	0	1	8	6	0	0
x1	0	1	1	1	1	1	0.25	9.25	1	1
x2	1	0	0	1	0	1	3	2.25	0	0
x3	-1	1	1	1	1	0	-7.87	13.12	1	1
x4	1.5	2	1	1	1	1	12.68	12.68	1	0

- Localizes (**N1-N2**) with weight **-1.5**

Example: Intermediate-layer Repair

- **Repair**

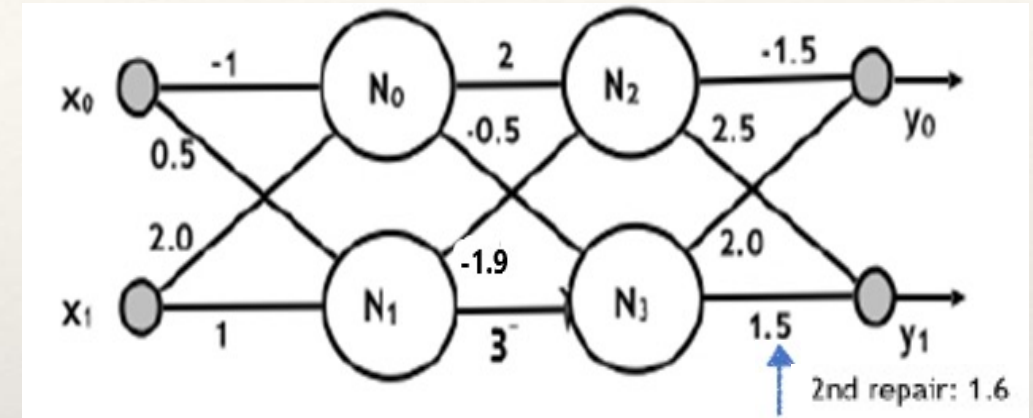
- Run the code using SPF to collect constraints for the failing test and passing tests
 - The weight of N1-N2 has a symbolic delta value added to it ($-1.5 + \delta$)
 - The repair constraint for failing test requires output of $N_2 \leq 0$
 - The repair constraints for all passing tests retain their behavior on N_2
- Z3 invoked to find a solution for delta that satisfies the repair constraints for failing and passing tests
- Changing the weight of N1-N2 from **-1.5** to **-1.9** changes the activation pattern for (N2; N3) to (off, on) on the failing input, as well as changes the output label to 0
- Repair retains existing behavior on the passing tests



	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

Example: Last-layer Repair

- Consider input $X_5 = [0.6; 1.0]$, misclassified to “0” (ideal is “1”)
- There isn’t a specific activation pattern for correct behavior for label 1, modify the behavior of last layer (decision-boundaries)
- Localization:**
 - Identify the edge in the last layer that has more impact on the behavior of failing tests than passing tests
 - Localizes edge **N3-Y1** with weight **1.5**
- Repair:**
 - Use SPF to collect repair constraints for X_5 with symbolic delta added to weight of **N3-Y1** and requiring $Y1 > Y0$, and constraints for passing tests
 - Z3 is used to find solution for delta; changing N3-Y1 to **1.6** flips the output for failing test while retaining the behavior for all passing tests

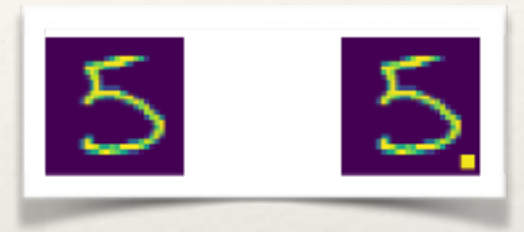


	x0	x1	N0	N1	N2	N3	y0	y1	class	ideal
X0	1	1	1	1	0	1	8	6	0	0
X1	0	1	1	1	1	1	0.25	9.25	1	1
X2	1	0	0	1	0	1	3	2.25	0	0
X3	-1	1	1	1	1	0	-7.87	13.12	1	1
X4	1.5	2	1	1	0	1	13.3	10.5	0	0
X5	0.6	1	1	1	1	1	5.91	5.62	0	1
after repair:	0.6	1	1	1	1	1	5.91	5.95	1	1

Experiments

- Three scenarios
 - Scenario 1: Improving overall accuracy of the model for all types of failures
 - Scenario 2: Fixing backdoor attacks
 - Scenario 3: Improving adversarial robustness
- Two common image-classification benchmarks
 - MNIST: 10 layer CNN, CIFAR-10: 15 layer CNN
 - Both networks include convolutional layers, maxpool layers, ReLU activation layers and softmax
- Research Questions
 - Is NNRepair effective for all three scenarios?
 - How do intermediate layer and last layer repair compare?
 - What is the inference time overhead?

Backdoor Attacks



The first image classified as 5, second image mis-classified to 7(poison target)

Adversarial Attacks



The first image classified as 5, second image mis-classified to another label

Results

- NNRepair improves accuracy in all three scenarios
 - Scenario 1: Improving accuracy for all failure types
 - Small amount (MNIST: 0.20%, CIFAR-10: 0.16%)
 - Effectiveness decreases as the accuracy of the original model increases, better than existing state-of-the-art for this difficult problem (MNIST-HQ: NNRepair: 0.02%, MODE: -0.40%,)
 - Scenario 2: Fixing backdoor attacks
 - Very good improvement in accuracy on poisoned inputs (MNIST: 45.56%, CIFAR-10: 3.77%) without decreasing accuracy on normal inputs
 - Scenario 3: Improving adversarial robustness
 - Good improvement in accuracy on adversarial inputs (MNIST: 10.40%, CIFAR-10: 0.34%) without decreasing the accuracy on normal inputs
- Intermediate layer vs Last layer repair
 - Intermediate layer seems to help more with improving overall accuracy and last layer seems better at repairing specific vulnerabilities
 - Pattern-based repair helps preserve behavior on normal inputs better
- While the inference time for the last-layer repair is comparable with the original model, the inference time for an intermediate-layer repair is expensive (F1-based filtering helps save time)

Conclusion

- Traditional program analysis techniques such as the SPF framework hold great potential in enabling debugging and verification of Neural Network models
- In this talk, we discussed a technique that performs constraint-based repair of faulty DNN models which precisely corrects faulty behavior while preserving good behavior
- NNRepair was shown to improve overall accuracy of the network and also improve the robustness of the model to vulnerabilities such as adversarial and backdoor attacks

Questions

Thank You!

divgml@gmail.com

divya.gopinath@nasa.gov

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