Analysis and Testing for AI

Wei Le

November 3, 2023

AI systems

- deep learning libraries (framework)
  - fuzzing to find deep learning library bugs
  - empirical studies
- AI application code
  - empirical studies from stack overflow, github, kaggle
  - empirical studies from product code
- AI models (after training) with parameters
  - testing model (fuzzing, test criteria)
  - static analysis for the model
  - dynamic analysis for the models
  - model repair
  - differential analysis [4]
  - abstract interpretation: AI^2

AI special properties

- Robustness
  - adversarial robustness: designed to attack the model
  - natural robustness: defect images, raining
- Generalization: how the model generalize across different projects, tasks
- Explanability
- Fairness: given two people with exact same profile but different races, will both of them get the loan?

Real-world case: Testing for Autonomous Vehicles at Uber

- collect real-world data
- generate synthetic data based on real-world data, e.g., lack of collision scenarios
- constantly running testing for the Autonomous Vehicles system
  - can you recognize the background data
  - can you recognize moving object
  - can you predict the next movement of the moving object

Neural Networks Input and Output

- Input: 3 Dimensional inputs (e.g., colored images)
- Reshape the input to vectors
- Output: labels (e.g., is it 1, 2, ...9)
- Using massive amount of labeled data to "parameterize" neural network for a specific problem, which we call *models*, models then used like a classifier

Neural Networks Internal

- consists of sequences of layers, e.g., input/output layer, and hidden layers
- each layer consists of *neurons* (also called *perceptron*)
- *feed-forward*: the output of a neuron is not feedback to the previous layer
- *convolutional neural networks*: *fully connected layers*, *pooling layers* and *convolutional layers*

Challenges

- Robustness: are neural network vulnerable to *adversarial examples* – slightly perturbing an input classified correctly leads to mis-classification?
- Testing: How to test models so we know it is a good model and ready to go for application?
- Debugging: if the prediction is wrong, is it a problem of insufficient training data, implementation errors in networks, or it is an error expected by the algorithm?

Guiding Deep Learning System Testing using Surprise Adequacy

Testing neural network in the era of AI engineering:

- in machine learning community, we use cross-folding, separate training dataset, validation dataset (tuning), test dataset randomly for 3-10 times.
- moving to AI engineering, we need high quality products
- introducing software testing methodogy, e.g., test criterion, when we gain confidence that the model is ready to deploy

Some recent work on neural network test criteria

Are all the neurons activated? when the nerons are activated, are a range of values/boundary values covered?

- DeepXplore's Neuron Coverage (NC)
- Three Neuron-level Coverages (NLCs) introduced by Deep-Gauge [27]: k-Multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC), and Strong Neuron Activation Coverage (SNAC).
  - k-multisection neuron coverage: given a neuron n, the criterion measures how thoroughly the given set of test inputs T covers the range $[\text{low}_n, \text{high}_n]$
  - neuron boundary converage: how many tests cover the corner cases of (high_n, ∞), and $(-\infty, \text{low}_n)$
  - strong neuron activation coverage: how many corner cases w.r.t. the upper boundary value high_n has been covered

Surprise of an input

- after training, the neural network should be able to handle similar input
- how surprise a given input compared to the training set
- what is the metric to meausre the surprise

Surprise metric: Distance based (work for classification problem)

Activation trace:

Let $\mathbf{N} = \{n_1, n_2, \ldots\}$ be a set of neurons that constitutes a DL system $\mathbf{D}$, and let $X = \{x_1, x_2, \ldots\}$ be a set of inputs. We denote the activation value of a single neuron n with respect to an input x as $\alpha_n(x)$. For an ordered (sub)set of neurons, let $N \subseteq \mathbf{N}$, $\alpha_N(x)$ denote a vector of activation values, each element corresponding to an individual neuron in N: the cardinality of $\alpha_N(x)$ is equal to $|N|$. We call $\alpha_N(x)$ the Activation Trace (AT) of x over neurons in N. Similarly, let $A_N(X)$ be a set of activation traces, observed over neurons in N, for a set of inputs X: $A_N(X) = \{\alpha_N(x) \mid x \in X\}$. We

Surprise metric: Euclidian Distance of between two activation traces

- Computing euclidean distance between the AT of a new input x and ATs observed during training.
- Classification problem: the notion of boundary of each class of input
- Given a new input x, and a predicted class of the new input $c_x \in C$, we define the reference point x_a to be the closest neighbour of x that shares the same class. The distance between x and x_a follows from the definition:

$$x_a = \underset{\mathbf{D}(x_i)=c_x}{\operatorname{argmin}} \|\alpha_{\mathbf{N}}(x) - \alpha_{\mathbf{N}}(x_i)\|, \qquad (3)$$
$$dist_a = \|\alpha_{\mathbf{N}}(x) - \alpha_{\mathbf{N}}(x_a)\|$$

Surprise metric: Euclidian Distance of between two activation traces

Subsequently, from x_a, we find the closest neighbour of x_a in a class other than c_x, x_b, and the distance $dist_b$, as follows:

$$\begin{aligned} x_b &= \underset{\mathbf{D}(x_i) \in C \setminus \{c_x\}}{\operatorname{argmin}} \|\alpha_{\mathbf{N}}(x_a) - \alpha_{\mathbf{N}}(x_i)\|, \\ dist_b &= \|\alpha_{\mathbf{N}}(x_a) - \alpha_{\mathbf{N}}(x_b)\| \end{aligned} \tag{4}$$

Intuitively, DSA aims to compare the distance from the AT of a new input x to known ATs belonging to its own class, c_x, to the known distance between ATs in class c_x and ATs in other classes in $C \setminus \{c_x\}$. If the former is relatively larger than the latter, x would be a surprising input for class c_x to the classifying DL system $\mathbf{D}$. While there are multiple ways to formalise this we select a simple one and calculate DSA as the ratio between $dist_a$ and $dist_b$. Investigation of more complicated formulations is left as future work.

$$DSA(x) = \frac{dist_a}{dist_b} \tag{5}$$

Surprise metrics: Likelyhood based (optional)

- Two types of metrics to quantify the "surprise": DSA (distance based adequacy) and LSA (likelyhood based surprise adequecy)
- Kernel Density Estimation (KDE): estimating the probability density function of a given random variable. The resulting density function allows the estimation of relative likelihood of a specific value of the random variable
- LSA: probability density of each activation value in $A_N(T)$
- $\alpha_N L$: certain layer, neuro's satisfy the threshold

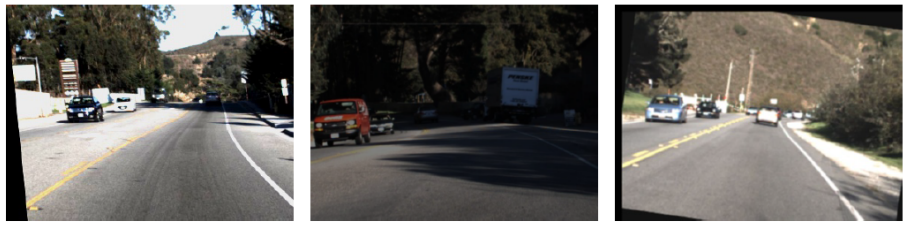
Surprise metrics: LSA

$$\hat{f}(x) = \frac{1}{|A_{N_L}(\mathbf{T})|} \sum_{x_i \in \mathbf{T}} K_H(\alpha_{N_L}(x) - \alpha_{N_L}(x_i)) \quad (1)$$

Adopting common approach of converting probability density to a measure of rareness [26], [39], we define LSA to be the negative of the log of density:

$$LSA(x) = -log(\hat{f}(x)) \quad (2)$$

Surprise metric LSA matches human perception



(a) Low LSA

(b) Medium LSA

(c) High LSA

Fig. 3: Synthetic images for Chauffeur model generated by DeepTest. Images with higher LSA values tend to be harder to recognise and interpret visually.

Research Questions

- RQ1. Surprise: Is SADL capable of capturing the relative surprise of an input of a DL system?
- RQ2. Layer Sensitivity: Does the selection of layers of neurons used for SA computation have any impact on how accurately SA reflects the behaviour of DL systems?
- RQ3. Correlation: Is SC correlated to existing coverage criteria for DL systems?
- RQ4. Guidance: Can SA guide retraining of DL systems to improve their accuracy against adversarial examples and synthetic test inputs generated by DeepXplore?

Experiment Setup

TABLE I: List of datasets and models used in the study.

Dataset	Description	DNN Model	# of Neuron	Synthetic Inputs	Performance
MNIST	Handwritten digit images composed of 50,000 images for training and 10,000 images for test.	A five layer ConvNet with max-pooling and dropout layers.	320	FGSM, BIM-A, BIM-B, JSMA, C&W.	99.31% (Accuracy)
CIFAR-10	Object recognition dataset in ten different classes composed of 50,000 images for training and 10,000 images for test.	A 12 layer ConvNet with max-pooling and dropout layers.	2,208	FGSM, BIM-A, BIM-B, JSMA, C&W.	82.27% (Accuracy)
Udacity Self-driving Car Challenge	Self-driving car dataset that contains camera images from the vehicle, composed of 101,396 images for training and 5,614 images for test. The goal of the challenge is to predict steering wheel angle.	Dave-2 [6] architecture from Nvidia.	1,560	DeepXplore's test input generation via joint optimization.	0.09 (MSE)
		Chauffeur [1] architecture with CNN and LSTM.	1,940	DeepTest's combined transformation.	0.10 (MSE)

Results: suprise of the adversial examples

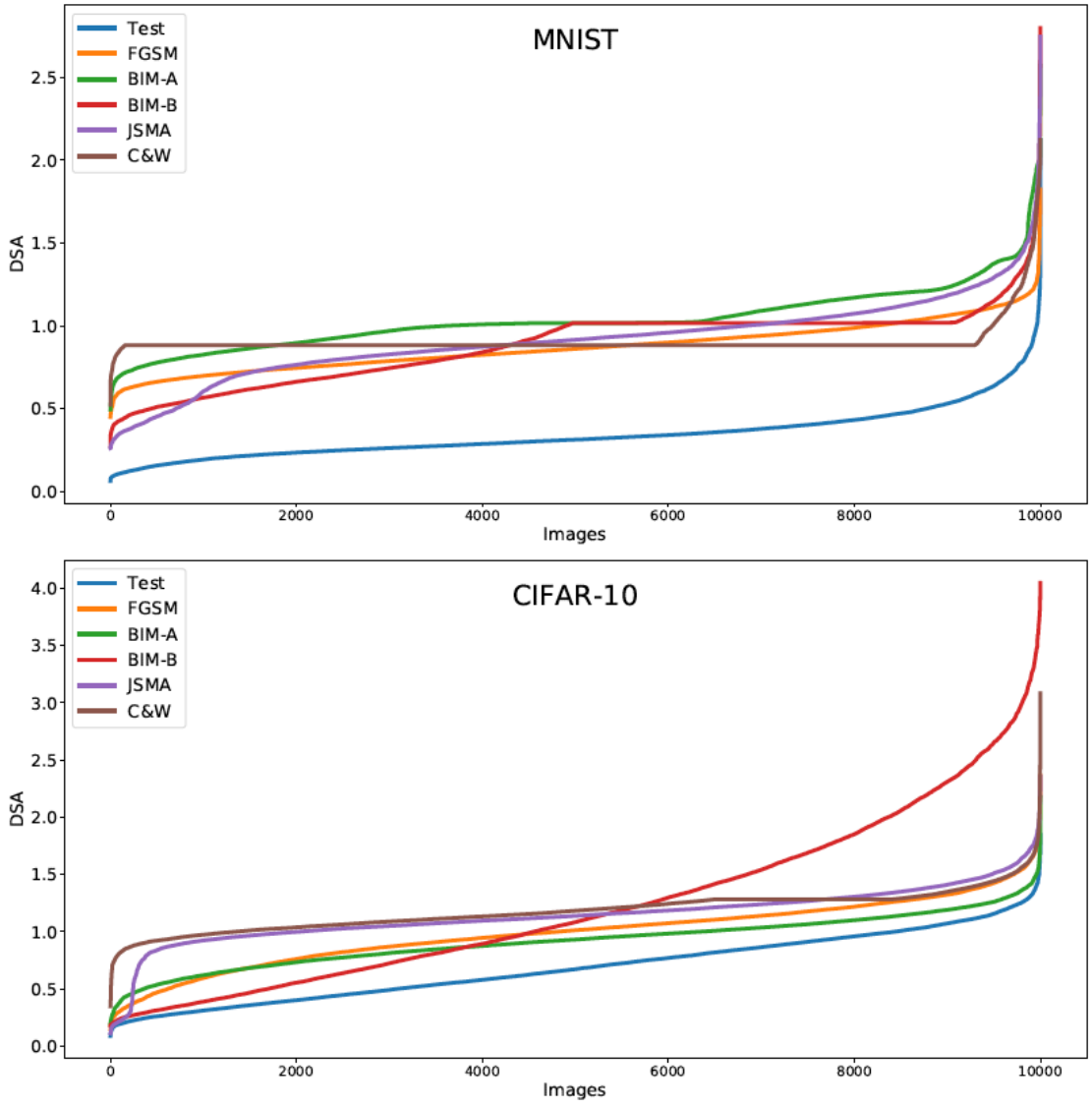


Fig. 4: Sorted DSA values of adversarial examples for MNIST and CIFAR-10.

Results: sensitive to layer selection

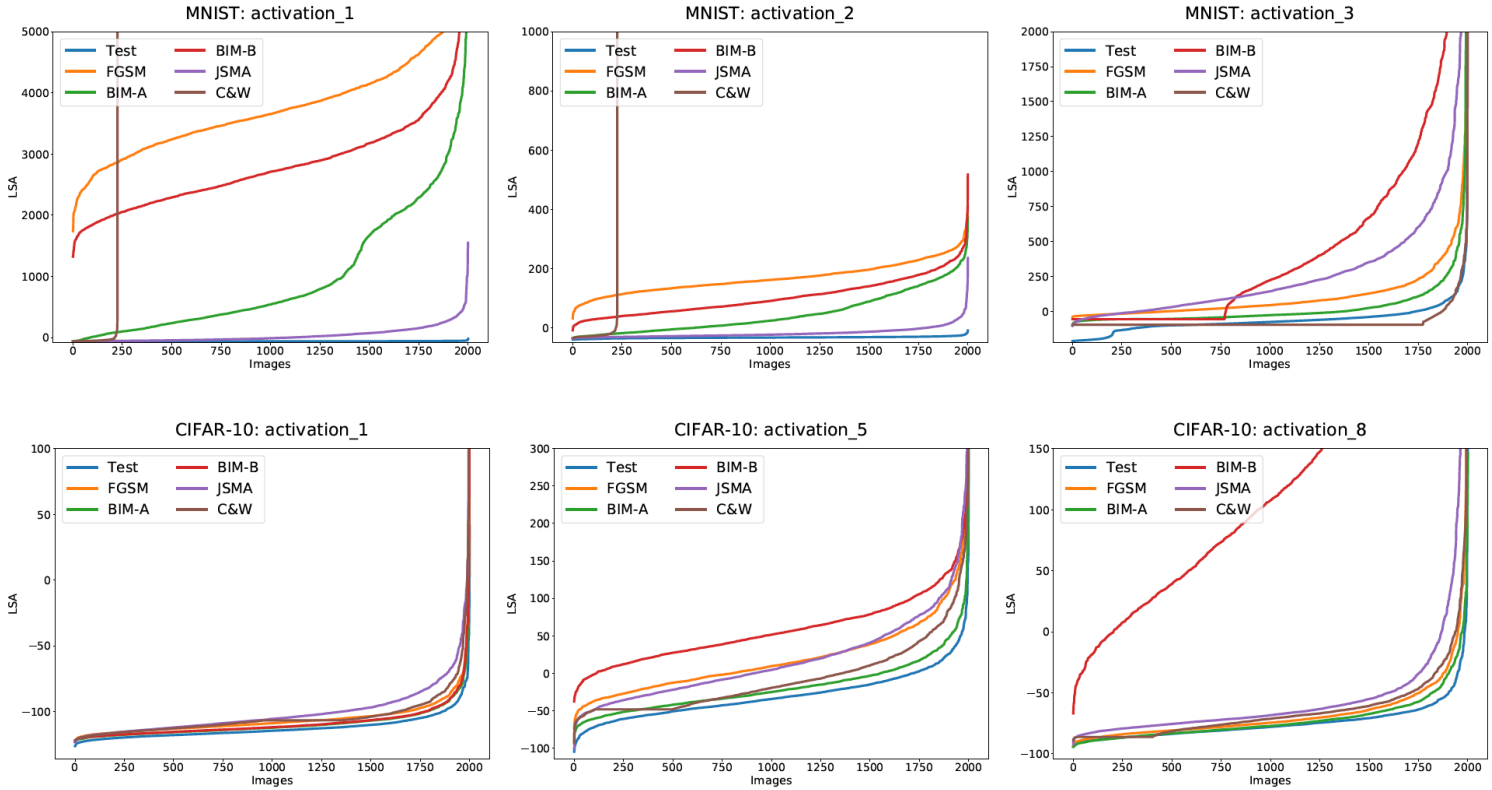


Fig. 5: Sorted LSA of randomly selected 2,000 adversarial examples for MNIST and CIFAR-10 from different layers

DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems

Dynamic analysis for RNN, building a DTMC (Discrete-Time Markov Chain) model based on RNN training

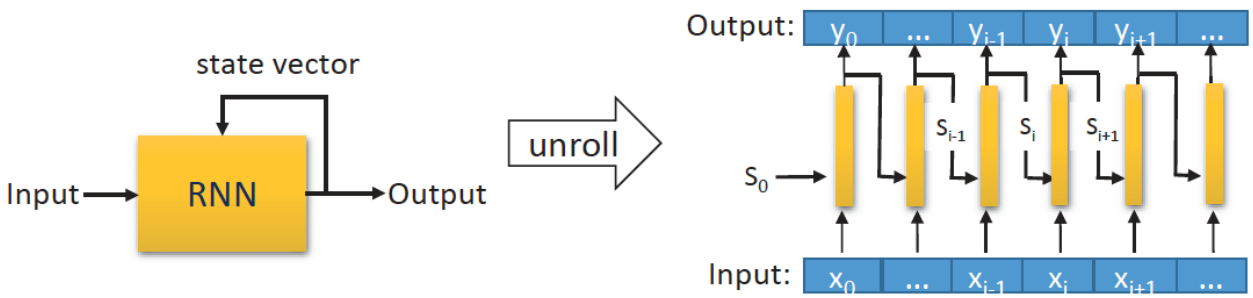
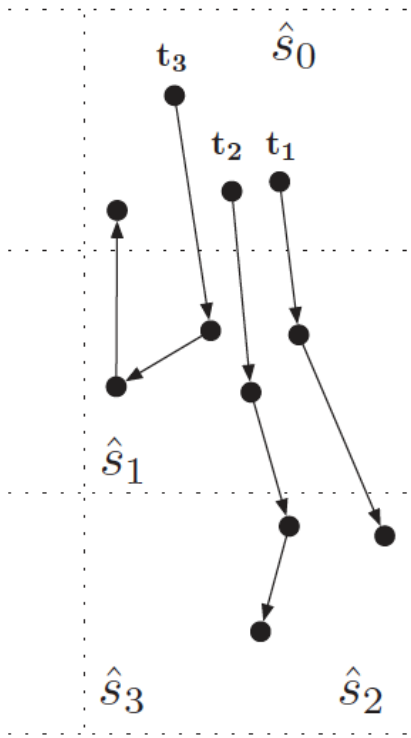


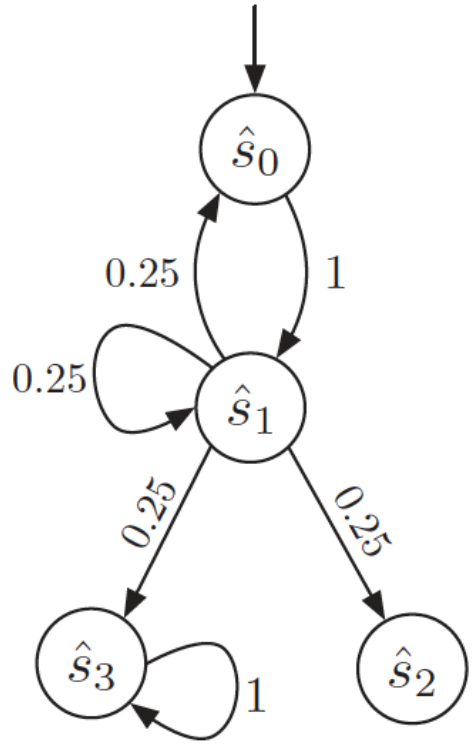
Figure 1: Architecture of a simple RNN.

DeepStellar: Model-Based Quantitative Analysis of Stateful Deep Learning Systems

An example DTMC: stateful, sequence

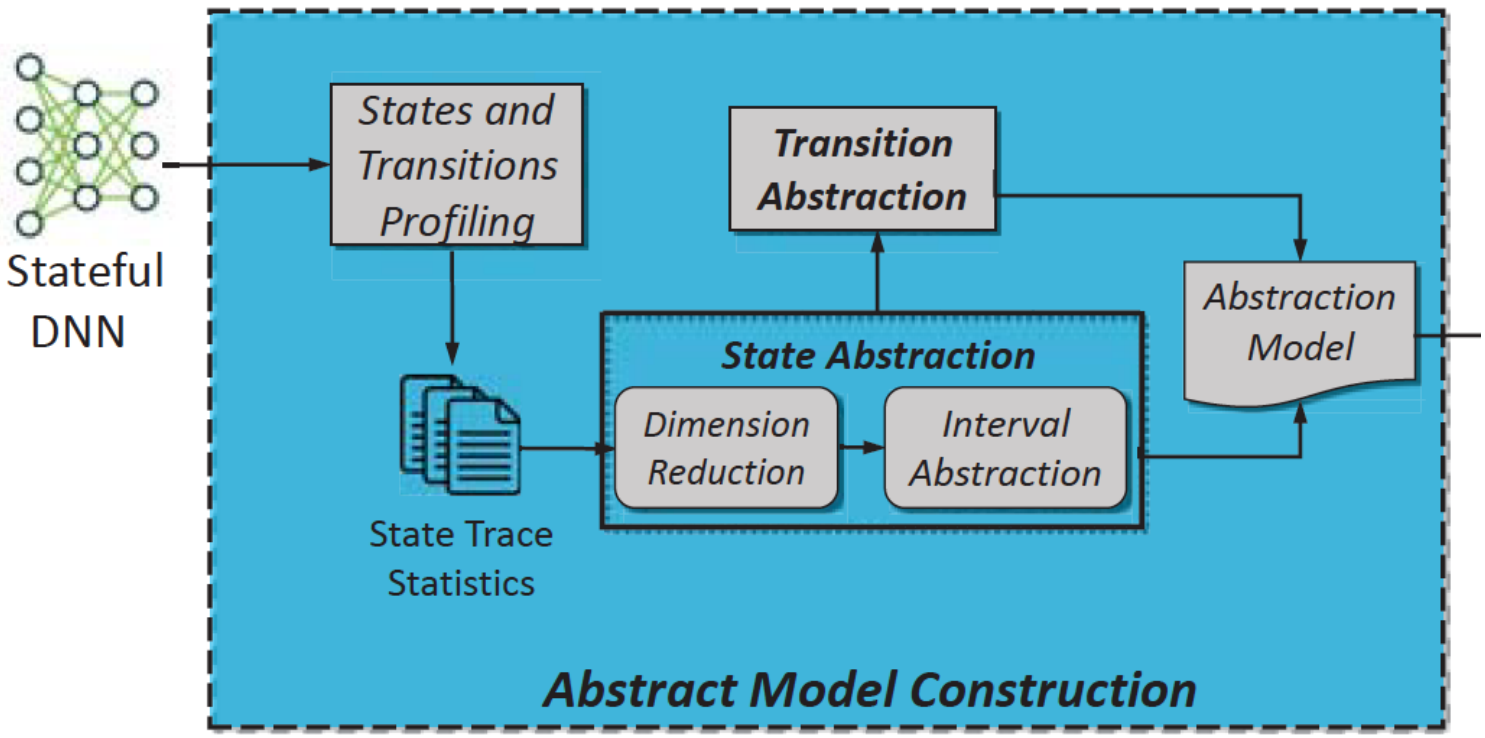


(a) Example concrete traces.



(b) DTMC abstraction.

DeepStellar: An overview approach



DeepStellar: Approach of building models

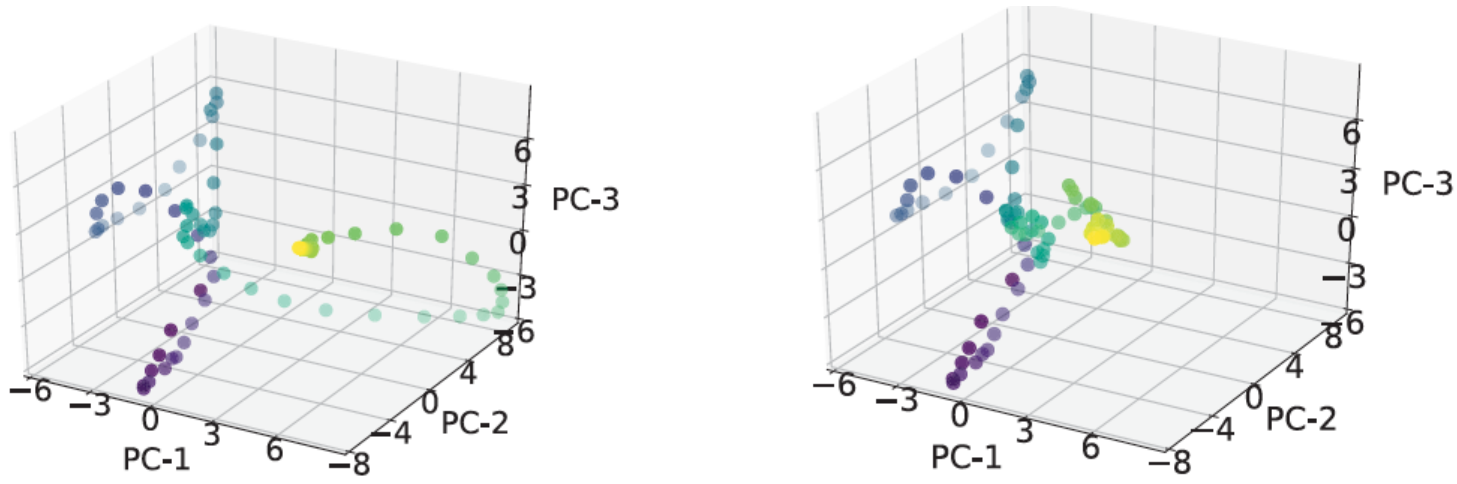
- State abstraction. each concrete state is represented as a vector usually in high dimension Intuitively, an abstract state represents a set of concrete states which are close in space. (PCA)
- Transition : An abstract transition represents a set of concrete transitions which share the same source and destination abstract states.

DeepStellar: Trace similarity

transitions on the abstract model. Given an abstract model M and an input $\mathbf{x}$, we denote the set of abstract states and transitions covered by $\mathbf{x}$ as $\hat{\mathcal{S}}_{\mathbf{x}}$ and $\hat{\delta}_{\mathbf{x}}$. Then, the *state- and transition-based trace similarity metrics* for the two inputs $\mathbf{x}$ and $\mathbf{y}$ are defined based on the Jaccard indices of their states and transitions covered, respectively:

$$\text{STSIM}_{\text{M}}(\mathbf{x}, \mathbf{y}) = \frac{|\hat{\mathcal{S}}_{\mathbf{x}} \cap \hat{\mathcal{S}}_{\mathbf{y}}|}{|\hat{\mathcal{S}}_{\mathbf{x}} \cup \hat{\mathcal{S}}_{\mathbf{y}}|}, \quad \text{TTSIM}_{\text{M}}(\mathbf{x}, \mathbf{y}) = \frac{|\hat{\delta}_{\mathbf{x}} \cap \hat{\delta}_{\mathbf{y}}|}{|\hat{\delta}_{\mathbf{x}} \cup \hat{\delta}_{\mathbf{y}}|}.$$

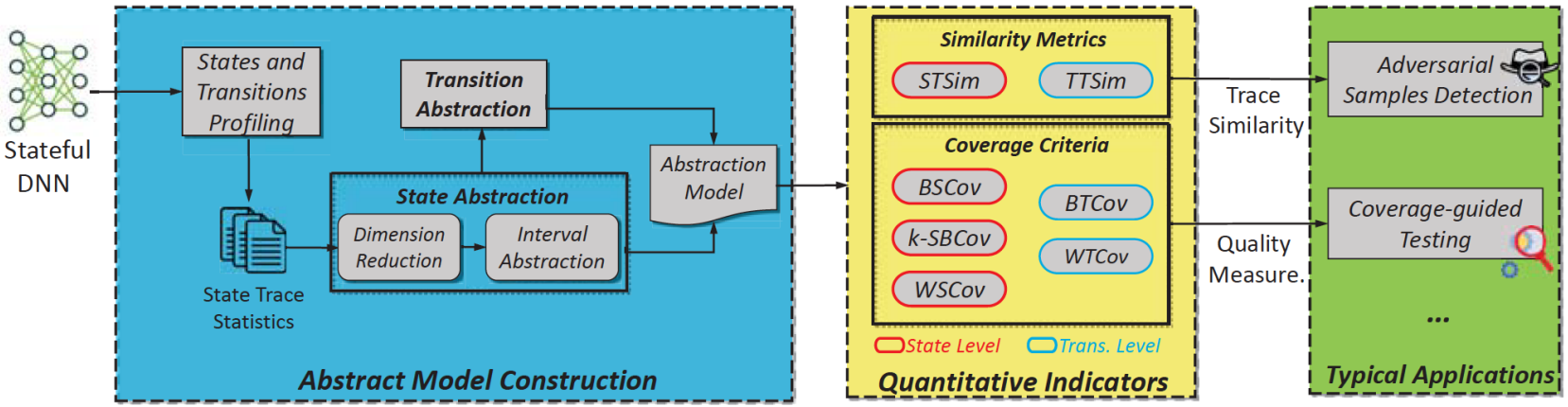
DeepStellar: Trace similarity



(a) "This book is about science." (b) "This book is about literature."

Figure 5: Visualization of concrete traces of two audio over an RNN-based ASR model with the PCA-3 abstraction.

DeepStellar: Applications



DeepStellar: Applications - Testing coverage criteria (optional)

State-level coverage

- Basic State Coverage.
- Weighted State Coverage.
- n-Step State Boundary Coverage.

Transition-level coverage

- Basic Transition Coverage
- Weighted Transition Coverage

DeepStellar: Applications - Coverage-Guided Testing of RNNs

Fuzzing like idea:

- In each run, it selects a seed (i.e., test case) from the queue and generates multiple mutants. A mutant is an adversarial sample if it is predicted incorrectly by the network.
- Otherwise, if the mutant improves the coverage, it is then retained as an interesting seed and added back to the queue.

DeepStellar: Applications - Adversarial Sample Detection for RNNs (optional)

Algorithm 1: Training an Adversarial Detection Classifier

input : D: RNN-based DL system, M: Abstract model of D
output : C: A classifier for detecting adversarial samples

Prepare benign set B, adversarial set A and reference set R;
$dis_b \leftarrow \emptyset$;
for $b \in B$ **do**
    $R' \leftarrow select(R, b)$;
    $vec \leftarrow \emptyset$;
    for $r \in R'$ **do**
        $(r_1, state\_vec1) \leftarrow predict(R, r)$;
        $(r_2, state\_vec2) \leftarrow predict(R, b)$;
        $j \leftarrow TraceSimilarity(state\_vec1, state\_vec2, M)$;
        $vec \leftarrow vec \bigcup \{j\}$;
    $d \leftarrow average(vec)$;
    $dis_b \leftarrow dis_b \bigcup \{d\}$;
Compute dis_a similar with dis_b;
$C \leftarrow LinearRegressionClassifer(dis_a, dis_b)$

DeepStellar: Results and findings

- Both state- and transition-level trace similarity metrics are capable of capturing the prediction difference even for slightly perturbed samples. Thus, trace similarity could be useful for detecting adversarial samples
- The coverage-guided testing is generally useful in terms of achieving higher coverage and guiding adversarial sample exploration. Among the three strategies, transition coverage-guided method achieves higher coverage, while state coverage-guided method uncovers more unique adversarial samples.

DeepGauge: Multi-Granuality Testing Criteria for Deep Learning Systems (2018) Optional

- each neuron computes an output based on an input
- each layer computes an output based on an input
- cover all possible output values
- design a family of test criteria for neuron level and layer level

Neuron Level Criteria

- k-multisection neuron coverage: given a neuron n, the criterion measures how thoroughly the given set of test inputs T covers the range [low_n, high_n]
- neuron boundary converage: how many tests cover the corner cases of (high_n, ∞), and ($-\infty$, low_n)
- strong neuron activation coverage: how many corner cases w.r.t. the upper boundary value high_n has been covered

Layer Level Criteria

- define "active neurons": for a given test input x and neuron n_1 and n_2, we say n_1 is more active than n_2 given x if the output of n_1 regarding x is larger than the output of n_2
- test data should uncover more active neurons
- top k neuron coverage: how many neurons of a layer has been the most active k neurons
- top k neuron patterns: how many top k neuron patterns are covered

Experimental Setup

- test dataset: MNIST and ImageNet
- include also adversarial test dataset

Findings

- the data set cover both main function region and corner cases, but cover the main function region more than corner cases
- the adversarial test dataset boost the coverage criteria
- lower region is more difficult to cover than the higher region

Further Reading

1. Guiding Deep Learning System Testing using Surprise Adequacy
2. AI^2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation
3. Deep Gauge: Multi-Granularity Testing Criteria for Deep Learning Systems
4. Brian McClendon's talk that covers testing for Autonomous Vehicles at Uber
5. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection
6. CRADLE: Cross-Backend Validation to Detect andLocalize Bugs in Deep Learning Libraries