

A Quantitative Analysis Framework for Recurrent Neural Network

Xiaoning Du*, Xiaofei Xie*, Yi Li*, Lei Ma[†], Yang Liu*, Jianjun Zhao[†]

*Nanyang Technological University, Singapore

[†]Kyushu University, Japan

Abstract—Recurrent neural network (RNN) has achieved great success in processing sequential inputs for applications such as automatic speech recognition, natural language processing and machine translation. However, quality and reliability issues of RNNs make them vulnerable to adversarial attacks and hinder their deployment in real-world applications. In this paper, we propose a quantitative analysis framework — *DeepStellar*— to pave the way for effective quality and security analysis of software systems powered by RNNs. *DeepStellar* is generic to handle various RNN architectures, including LSTM and GRU, scalable to work on industrial-grade RNN models, and extensible to develop customized analyzers and tools. We demonstrated that, with *DeepStellar*, users are able to design efficient test generation tools, and develop effective adversarial sample detectors. We tested the developed applications on three real RNN models, including speech recognition and image classification. *DeepStellar* outperforms existing approaches three hundred times in generating defect-triggering tests and achieves 97% accuracy in detecting adversarial attacks. A video demonstration which shows the main features of *DeepStellar* is available at: <https://sites.google.com/view/deepstellar/tool-demo>.

I. MOTIVATION

Over the past decades, we have witnessed the emergence and rapid development of deep learning (DL). DL has been successfully deployed in many real-life applications, including face recognition, automatic speech recognition (ASR) and autonomous driving, etc. However, due to the intrinsic vulnerability and the lack of rigorous verification, DL systems suffer from quality and security issues, such as the Alexa/Siri manipulation and the autonomous car accidents, which are introduced from both the development and deployment stages [1].

Due to the fundamentally different programming paradigm and logic representation from traditional software, existing quality assurance techniques can hardly be directly applied to DL systems. Recently, significant research efforts have been made on techniques specific to DL systems, including testing [2], [3], [4], [5], [6], verification [7], and adversarial sample detection [8]. The existing work mainly focus on the Feed-forward Neural Networks (FNNs), leaving the Recurrent Neural Networks (RNNs) untouched. FNNs are with a feed-forward design and take an input as a monolithic piece; each layer inside plays a fixed role in the feature extraction process. In contrast, RNNs consume a sequential input segment by segment and work in an iterative manner, as illustrated in Fig. 1a. In each iteration, the RNN processes an individual element, evolves into a new state, records and forwards the state information to the next iteration. This key feature enables

RNNs to handle data rich in temporal information, such as natural language texts, audios and videos. The fact that the roles of layers are interchangeable, makes the quantitative analysis of RNNs much more difficult, even to measure the behavioral differences when processing two samples.

To bridge the gap, we propose a general-purpose quantitative analysis framework, *DeepStellar*, which enables one to perform effective security and quality analyses for RNN-based DL systems. The core of *DeepStellar* is an abstraction of the unique hidden state space of RNNs, which models the internal behaviors of RNNs as Discrete-Time Markov Chain (DTMC). *DeepStellar* is *generic* to handle various RNNs architectures, including LSTM [9] and GRU [10], and *extensible* to develop customized analyzers and testing tools for RNNs. To demonstrate the effectiveness of *DeepStellar*, we developed an efficient test generation tool and effective adversarial sample detectors for RNNs, based on *DeepStellar*. We apply both tools on three real RNN models, including ASR and image classification, and observed promising results with hundreds of times more adversarial samples generated, and 97% accuracy in adversarial attack detection. *DeepStellar* facilitates developers, users and researchers of RNN-based DL systems, allowing them to have a better comprehension on the behaviors of RNNs, define quantitative measures, and develop more useful toolkit to make the systems more robust and secure.

II. THE *DeepStellar* FRAMEWORK

Fig. 1b shows an overview of *DeepStellar*, which includes three components: the abstract model construction module, and two applications – the adversarial sample detector (*DeepStellar*-ASD) and coverage-guided testing tool (*DeepStellar*-CGT). The abstraction module takes the target RNN and (a part of) its training data as inputs, and constructs an abstract model. *DeepStellar* provides APIs to access the state and transition information from the abstract model, and the abstract trace traversed by a sample. In this way, users are able to develop customized analyzers and testing tools for RNNs by extending *DeepStellar*, similar to *DeepStellar*-ASD and *DeepStellar*-CGT.

A. Abstract Model Construction

For an RNN, *DeepStellar* generates an abstract model (i.e., DTMC model) via three steps: 1) states and transitions profiling, 2) state abstraction, and 3) transition abstraction.

Step 1: Profiling. *DeepStellar* uses inputs from the training data to perform the profiling, as they can manifest the

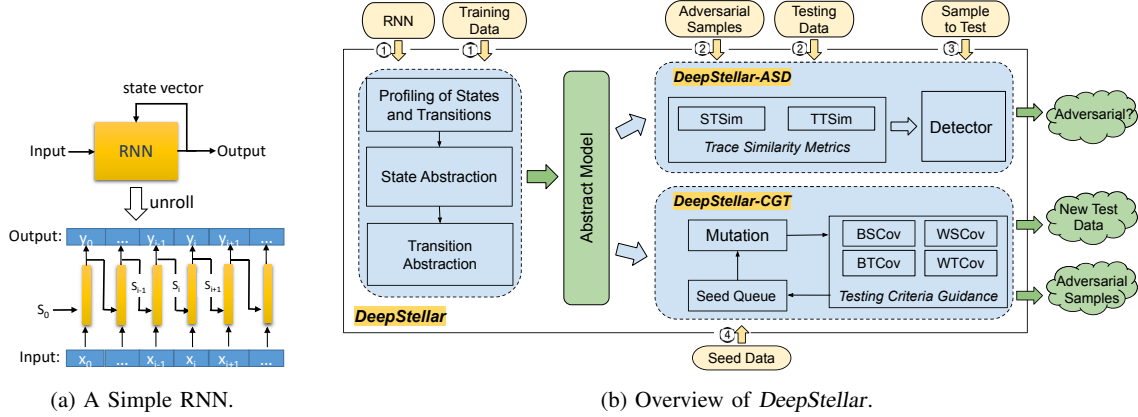


Fig. 1: Architecture of RNN and Overview of *DeepStellar*.

characteristics of a trained RNN model. Via executing the RNN on an input, a trace consisting of state vectors is generated. In this way, from the training data, we obtain a set of traces recording the concrete states visited and the concrete transitions taken during the training stage. Based on the concrete states and transitions, we then perform the state and transition abstraction.

Step 2: State Abstraction. We first group all the concrete state vectors and apply Principle Component Analysis (PCA) to identify the first k principle components which distinguish the vectors to the most extend. Then, we project the concrete state vectors onto the k -dimensional component basis. This is to reduce the dimension of the state space and improve computation efficiency, since concrete state vectors may come with very high dimension. Next, we perform an interval abstraction by dividing each axis into m equal-length intervals, and the k -dimensional space are split into m^k regular grids [11]. Finally, we map concrete states falling into the same grid to the same abstract state. Note that the precision of state abstraction is configurable by the parameters (k, m) .

Step 3: Transition Abstraction. We abstract the concrete transitions based on the state abstraction. An abstract transition represents a set of concrete transitions which share the same source and destination abstract states. For each abstract transition t , we calculate its transition probabilities by taking the number of concrete transitions, that mapped to t , over the number of all outgoing concrete transitions that share the same source abstract state as t .

B. Applications

Based on the abstract model, we developed two applications for 1) adversarial sample detection (*DeepStellar*-ASD), and 2) coverage-guided testing (*DeepStellar*-CGT) of RNNs. Here, we give a brief description about their designs and usage.

1) *Adversarial Sample Detection*: Adversarial samples are able to fool RNNs with human-imperceptible perturbations. We speculate that there exist some abnormal behaviors during the process of predicting over the adversarial samples, compared with benign samples. For example, when a perturbed *panda* picture is recognized as a *gorilla*, the behavioral trace of RNN is supposed to be different from that of a real *gorilla*,

because visible differences exists between these two pictures. With *DeepStellar*, we can capture finer-grained behaviors of RNNs and even to measure the differences between samples with minor perturbations. Specifically, we designed two trace similarity metrics, and developed an algorithm to detect adversarial samples based on them.

Trace Similarity Metrics. The two similarity metrics (refer to [12] for the detailed definitions) are from both the state and transition levels, i.e., *state-based trace similarity* (SBTSIM) and *transition-based trace similarity* (TBTSIM). They are derived to compare the trace similarity between samples from the Jaccard index of their abstract states or transitions covered. Given two samples, we can leverage *DeepStellar* to get the set of states or transitions covered by each, and calculate the Jaccard index between the two sets.

Usage. *DeepStellar*-ASD takes a set of adversarial samples and a (partial) set of testing data (highlighted with step ② in Fig. 1b) as inputs, and outputs a detector which checks whether a given sample (step ③) is adversarial.

To check whether a given sample (step ③) is adversarial, *DeepStellar*-ASD first generates *reference* samples which have the same the prediction output as the target. The trace similarity between the original sample and its reference samples are used as an indicator to differentiate between benign and adversarial samples. Basically, the similarity between a benign sample and its references is larger than that of an adversarial sample. For the *panda* as *gorilla* example, its reference samples are all *gorilla*, thus the similarity would be much larger than that between *gorilla* pictures. For the image classification task, we get 50 reference samples from the training data based on the label of the original sample. For the ASR models, we use off-the-shelf text-to-speech engines to generate a standard audio from the transcript of the original sample.

2) *Coverage-Guided Testing*: With the profiling of samples from training data, *DeepStellar* is able to capture the major behavioral space and transitions in normal cases. The objective of testing is to systematically generate test cases, which may also explore regions outside the major behavioral space. We derive a set of coverage criteria to facilitate the guided testing

TABLE I: Studied subject model information.

| Subject Model | Kernel RNN | | # Traina. | Acc. (%) | |
|------------------|------------|------------------|---------------------|----------|-------|
| | Type | State vec. shape | Param. | Train. | Test. |
| DeepSpeech 0.1.1 | Bi-LSTM | (None, 4096) | 122x10 ⁶ | - | - |
| MNIST-LSTM | LSTM | (None, 128) | 81,674 | 99.69 | 98.66 |
| MNIST-GRU | GRU | (None, 128) | 61,578 | 99.70 | 98.61 |

of RNNs, with the aim to improve the testing dataset adequacy and uncover more defects.

Coverage Criteria. For the coverage criteria, *basic state coverage* (BSCOV), *weighted state coverage* (WSCOV), *basic transition coverage* (BTCOV) and *weighted transition coverage* (WTCOV) are designed to quantify the portion (or weighted portion considering the visiting probabilities) of abstract states/transitions visited by any test dataset. They indicate how adequately the internal states/transitions are exercised.

Usage. *DeepStellar*-CGT begins with a set of initial seeds (step ④ in Fig. 1b), and returns a set of augmented test data and a group of adversarial samples discovered. Firstly, the initial seed data are put into a seed queue, then *DeepStellar*-CGT iterates to increase a chosen coverage criteria of the seed queue with sample mutation and selection. In each iteration, we select a seed from the queue and generate a series of mutants. Mutation strategies to augment the test data are designed for image, natural language, and audio, respectively. The mutation follows metamorphic relations such that the perturbations applied are minor and would not change the truth label. For the newly generated samples, if they are adversarial, we keep them in the adversarial sample group. Otherwise, we check their contribution to the specific coverage criteria, and retain the test cases that cover new states or transitions. *DeepStellar*-CGT iteratively continues the above steps until the given time budget exhausts.

III. IMPLEMENTATION AND FEATURES

We have implemented *DeepStellar* in Python based on Keras-2.2.4 and TensorFlow-1.11. Its command line interface provides three key features: (1) RNN abstract model construction, (2) RNN adversary example detection, and (3) RNN coverage-guided testing. These features can help end users to perform the quality and security analysis of RNN directly. Furthermore, *DeepStellar* provides a comprehensive set of APIs to assist further application development for RNN analysis: (1) APIs for retrieving the detailed information of the abstract model, and (2) APIs to visualize the runtime traces of RNN predictions.

To validate the practical value of *DeepStellar*, we applied *DeepStellar*-ASD on detecting adversarial samples generated by state-of-the-art attack techniques. We further applied *DeepStellar*-CGT on image classification RNN models trained with MNIST dataset, and generated a number of adversarial samples effectively. The results demonstrate that *DeepStellar* is useful on revealing and evaluating the defects of RNN-based DL systems. With these engineering efforts and experimental results, we believe that *DeepStellar* is scalable and effective in attacking and defending real-world RNNs.

TABLE II: AUROC results (%) of trace similarity based adversarial detection by configurations.

| Config. | DeepSpeech-0.1.1 | | | MNIST-LSTM | | | | | |
|---------|------------------|--------------|--------------|--------------|--------------|--------------|--------------|--------------|-------|
| | (k, m) | STSIM | TTSIM | FSGM | | BIM | | DeepFool | |
| | | | | STSIM | TTSIM | STSIM | TTSIM | STSIM | TTSIM |
| (2, 40) | 81.00 | 50.00 | 79.95 | 88.01 | 77.9 | 80.77 | 76.82 | 81.86 | |
| (2, 80) | 70.25 | 50.00 | 90.54 | 96.55 | 82.38 | 92.12 | 84.43 | 92.46 | |
| (3, 5) | 69.41 | 85.40 | 86.95 | 85.31 | 84.71 | 83.25 | 82.42 | 80.15 | |
| (3, 10) | 89.26 | 85.19 | 90.05 | 89.74 | 86.18 | 85.28 | 85.23 | 83.78 | |
| (3, 20) | 85.25 | 50.00 | 89.4 | 92.62 | 84.63 | 84.72 | 83.53 | 85.46 | |
| (3, 40) | 50.49 | 50.00 | 90.56 | 93.79 | 85.97 | 86.62 | 84.05 | 87.47 | |
| (3, 80) | 50.00 | 50.00 | 96.63 | 93.64 | 92.97 | 89.75 | 93.44 | 90.43 | |

IV. EVALUATION

We demonstrate the usefulness of the *DeepStellar* applications, on three RNNs from domains of speech recognition and image classification. Details of the subject models can be found in Table I. The ASR model used is Mozilla’s implementation of DeepSpeech 0.1.1, which is of industrial-level performance. For image classification, we trained two RNNs with the LSTM and GRU architectures over the MNIST dataset, respectively. Both models achieve over 98% test accuracy. They covered different RNN architectures including LSTM, bidirectional LSTM and GRU. The largest model, DeepSpeech 0.1.1, contains over one hundred millions of trainable parameters, and the state vectors are in 4,096 dimensions. In addition, we evaluated the usefulness of the proposed quantitative measures on two application tasks, namely detecting adversarial examples and coverage guided testing for RNNs. More details on the experiment setup and results can be found in [12].

(1) Adversarial Sample Detection. Adversarial samples for each model are generated with state-of-the-art attack methods. We use the C&W audio attack [13] to generate adversarial audios for DeepSpeech 0.1.1, and use *FGSM*, *BIM* and *DeepFool* to generate adversarial samples for image classification models. For each RNN, we generate 13 abstract models under multi-grained (k, m) abstraction configurations. Finally, for each attack method, we generated 5,000 adversarial samples and randomly select the same number of benign samples. With these samples, we separately sampled the SBTSIM and TBTSIM measures, and trained a linear regression classifier for detecting adversarial examples. We show several AUROC results in Table II for two RNNs – DeepSpeech 0.1.1 and MNIST-LSTM, and the best detection accuracy hits 89% and 97%. The results demonstrate that models with different abstractions can achieve different detection accuracy. *DeepStellar* is useful on detecting adversarial examples by selecting suitable abstract models.

(2) Coverage-guided Testing. We evaluated the usefulness of *DeepStellar*-CGT on coverage increase and the number of adversarial examples with the BSCov guidance and BTCov guidance. We selected the abstract model with the configuration ($k=3, m=10$). Results for the MNIST models are shown in Table III. For each coverage guidance, we analyze the coverage increase on all testing criteria (i.e., the first column). Column “Seed” shows the initial coverage from the initial seeds. The coverage results from state- and transition- coverage guidance

TABLE III: Results of coverage and unique adversarial samples by different testing strategies.

| Criteria (%) | MNIST-LSTM | | | | MNIST-GRU | | | | | |
|--------------|------------|---------------|--------------|-------------|-----------|---------|---------------|--------------|--------|-------|
| | Seed | S-Guid. | T-Guid. | Ran. DeepT. | Seed | S-Guid. | T-Guid. | Ran. DeepT. | | |
| BSCov | 54.59 | 97.78 | 97.78 | 86.23 | 65.35 | 63.20 | 95.87 | 96.04 | 88.78 | 68.98 |
| WSCov | 96.44 | 99.99 | 99.99 | 99.66 | 98.04 | 97.03 | 99.98 | 99.98 | 99.80 | 98.05 |
| BTCov | 15.13 | 53.43 | 96.43 | 73.88 | 26.23 | 14.42 | 42.80 | 93.89 | 71.82 | 21.32 |
| WTCov | 77.80 | 94.81 | 99.90 | 98.02 | 85.12 | 63.40 | 88.78 | 99.69 | 96.95 | 72.34 |
| #Unique Adv. | - | 87,596 | 41,614 | 2,219 | 300 | - | 69,777 | 35,228 | 19,738 | 244 |

are shown under Column “S-Guid.” and “T-Guid.”, respectively. For comparison, we included a random testing without coverage guidance (Column “Ran.”) and an existing neuron coverage guided testing tool for RNN based on unrolling [3] (Column “DeepT.”). We can see that *DeepStellar*-CGT outperforms all the baseline approaches on both the coverage criteria increase and the generation of adversarial samples. Specifically, *DeepStellar*-CGT discovers 3 to 40 times more adversarial samples than random testing, and around 300 times more than DeepTest.

V. RELATED WORK

Abstraction of RNN. There exist some pioneer studies on the abstraction techniques for RNNs, but they mostly used Finite State Automaton (FSA) to capture RNNs’ internal dynamics, which lacks the transition probability distributions as in DTMC. In the literature, the existing studies are mostly concerned with the strategy for the internal state space partition, which is one of the most important techniques in the abstraction. Proposed partitioning strategies include equal division of each dimension for regular grids [11], unsupervised classification algorithms such as k -means and its variants [14], [15] and dynamic partition schemes with kernel algorithms [7]. However, they all suffer from the scalability problem when applied to real-world RNNs where the internal state space can be extremely large and in high dimension. Instead, *DeepStellar* employs PCA for a much cheaper abstraction.

RNN Adversarial Example Detection. Adversarial example detection for RNNs is still at an early stage. Techniques [8], [16], [17] specially designed for feedforward neural networks cannot be applied to RNNs. The softmax probability based approach [16] could possibly be used to detect RNN adversarial examples for classification problem only. As far as we know, *DeepStellar*-ASD is the first tool specifically designed to detect adversarial samples for RNNs with sequential output.

Deep Learning Testing. Recently, DNN testing has been widely studied including the study on testing criteria [2], [4] and testing tool [2], [3], [5], [6], [18]. However, most of them mainly focus on feed forward neuron networks. DeepTest [3] can be used on RNNs by unrolling. However, it unrolls the RNN with fixed iterations and is not scalable. TensorFuzz [18] can be used on RNN as it only considers the output of the logits layer. However, it lacks internal analysis of the RNNs.

VI. CONCLUSION AND FUTURE WORK

DeepStellar provides a fundamental infrastructure supporting versatile analyses of RNNs. We demonstrate two typical

applications of *DeepStellar* with a test generation tool and adversarial sample detector. For future work, we intend to augment the abstraction with transition labels, i.e., the input triggering a transition, and abstraction techniques can handle inputs from continuous space. We are also interested in investigating what and how critical properties can be verified over the abstract model to unveil deeply buried defects in RNNs.

Acknowledgment. This research was supported (in part) by Singapore National Research Foundation, under its National Cybersecurity R&D Program (Award No. NRF2018NCR-NCR005-0001) and National Satellite of Excellence in Trustworthy Software System (Award No. NRF2018NCR-NSOE003-0001), and NTU research grant NGF-2019-06-024, and JSPS KAKENHI Grant 19K24348, 19H04086, and Qdaijump Research Program NO.01277. Also, we would like to thank Nvidia for the hardware support of GPU cards and servers, and Mr. Yun Tang for the great help during preparation of the demo.

REFERENCES

- [1] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, “An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms,” in *ASE*, 2019.
- [2] K. Pei, Y. Cao, J. Yang, and S. Jana, “Deepxplore: Automated whitebox testing of deep learning systems,” in *SOSP*, 2017, pp. 1–18.
- [3] Y. Tian, K. Pei, S. Jana, and B. Ray, “Deeptest: Automated testing of deep-neural-network-driven autonomous cars,” in *ICSE*, 2018, pp. 303–314.
- [4] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu *et al.*, “Deepgauge: Multi-granularity testing criteria for deep learning systems,” 2018, pp. 120–131.
- [5] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, “Deephunter: A coverage-guided fuzz testing framework for deep neural networks,” in *ISSTA*, 2019, pp. 146–157.
- [6] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, “Diffchaser: Detecting disagreements for deep neural networks,” in *IJCAI*, 2019, pp. 5772–5778.
- [7] G. Weiss, Y. Goldberg, and E. Yahav, “Extracting automata from recurrent neural networks using queries and counterexamples,” in *ICML*, 2018, pp. 5244–5253.
- [8] J. Wang, G. Dong, J. Sun, X. Wang, and P. Zhang, “Adversarial sample detection for deep neural network through model mutation testing,” in *ICSE*, 2019, pp. 1245–1256.
- [9] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [10] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, “On the properties of neural machine translation: Encoder-decoder approaches,” *arXiv preprint arXiv:1409.1259*, 2014.
- [11] C. W. Omlin and C. L. Giles, “Constructing Deterministic Finite-State Automata in Recurrent Neural Networks,” *JACM*, vol. 43, no. 6, pp. 937–972, 1996.
- [12] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, and J. Zhao, “Deepstellar: Model-based quantitative analysis of stateful deep learning systems,” in *FSE*, 2019, pp. 477–487.
- [13] N. Carlini and D. Wagner, “Audio Adversarial Examples: Targeted Attacks on Speech-to-Text,” in *SPW*, 2018, pp. 1–7.
- [14] A. L. Cechin, D. R. P. Simon, and K. Stertz, “State Automata Extraction from Recurrent Neural Nets Using k-Means and Fuzzy Clustering,” in *SCCC*, 2003, pp. 73–78.
- [15] B.-J. Hou and Z.-H. Zhou, “Learning with Interpretable Structure from RNN,” oct 2018. [Online]. Available: <http://arxiv.org/abs/1810.10708>
- [16] D. Hendrycks and K. Gimpel, “A baseline for detecting misclassified and out-of-distribution examples in neural networks,” *arXiv preprint arXiv:1610.02136*, 2016.
- [17] W. Xu, D. Evans, and Y. Qi, “Feature squeezing: Detecting adversarial examples in deep neural networks,” *arXiv preprint arXiv:1704.01155*, 2017.
- [18] A. Odena and I. Goodfellow, “Tensorfuzz: Debugging neural networks with coverage-guided fuzzing,” in *ICML*, 2019, pp. 4901–4911.