2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

Towards Improved Testing For Deep Learning

Jasmine Sekhon
University of Virginia
Charlottesville, VA USA
js3cn@virginia.edu

Cody Fleming
University of Virginia
Charlottesville, VA USA
cf5eg@virginia.edu

***Abstract*—The growing use of deep neural networks in safety-critical applications makes it necessary to carry out adequate testing to detect and correct any incorrect behavior for corner case inputs before they can be actually used. Deep neural networks lack an explicit control-flow structure, making it impossible to apply to them traditional software testing criteria such as code coverage. In this paper, we examine existing testing methods for deep neural networks, the opportunities for improvement and the need for a fast, scalable, generalizable end-to-end testing method. We also propose a coverage criterion for deep neural networks that tries to capture all possible parts of the deep neural network's logic.**

***Index Terms*—deep neural networks, whitebox testing, coverage criterion**

I. Introduction

Deep Neural Networks, or DNNs, are increasingly being used in diverse applications owing to their ability to match or exceed human level performance. The availability of large datasets, fast computing methods and their ability to achieve good performance has paved way for DNNs into safety-critical avenues such as autonomous car driving, medical diagnosis, security, etc. The safety-critical nature of such applications makes it imperative to adequately test these DNNs before deployment. However, unlike traditional software, DNNs do not have a clear control-flow structure. They learn their decision policy through training on a large dataset, adjusting parameters gradually using several methods to achieve desired accuracy. Consequently, traditional software testing methods like functional coverage, branch coverage, etc. cannot be applied to DNNs, thereby challenging their use for safety-critical applications.

A lot of recent work, discussed in III, has looked into developing testing frameworks for DNNs. These methods suffer from certain limitations, as discussed in IV. In our work, we intend to make an effort to overcome these limitations and build a fast, scalable, efficient, generalizable testing method for deep neural networks. In V, we propose a coverage criterion for feed forward deep neural networks that tries to capture the DNN logic to a greater extent by incorporating inter-layer and intra-layer relationships.

II. Background

Deep neural networks are neural networks with multiple hidden layers between the input and output layers. Unlike traditional software programs, where the program logic has to be manually described by the programmer, deep neural networks are capable of learning rules by training on a large dataset. Today, DNNs are used in easy to complex tasks, such as image classification [10], medical diagnosis and end-to-end driving in autonomous cars [1]. The safety-critical nature of such applications makes it important to assure correctness, to avoid fatally incorrect behavior and obtain performance benefits from DNNs safely.

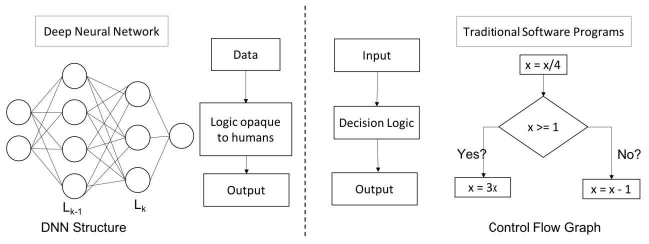

Fig. 1: The internal logic of a deep neural network is opaque to humans, as opposed to the well laid out decision logic of traditional software programs.

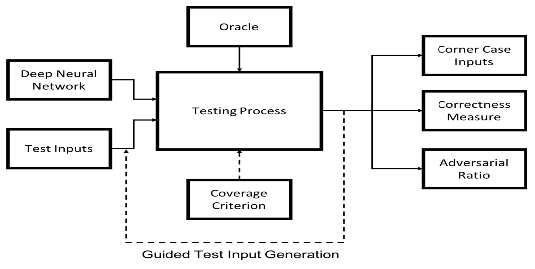

Fig. 2: A high-level representation of most existing DNN testing methods.

Traditional software testing methods fail when applied to DNNs because the code for deep neural networks holds no information about the internal decision-making logic of a DNN, as shown in Figure 1. DNNs learn their rules from training data and lack the control-flow structure present in traditional software programs. Therefore, traditional coverage criterion like code coverage, branch coverage, functional coverage, etc. cannot be applied to deep neural networks. A high-level representation of most existing whitebox testing methods for DNNs is shown in Figure 2. The inputs to the testing process are the DNN, the test inputs, and a coverage metric to ensure that all parts of the program logic have been tested. An oracle decides whether the behavior of the DNN is correct for the tested inputs. Further, a guided test input generation method may be used to generate test inputs that have *greater coverage* and which *uncover greater corner case behavior*. The output of existing testing methods is usually either a measure of system correctness or adversarial ratio.

III. Prior Literature

Testing methods for deep neural networks have normally followed a black-box approach, until recently, when DeepXplore [7] proposed the first white-box testing method for DNNs. The method proposed by [6] involves randomly searching around a given input for changes that cause misclassification. Many other approaches involve generating adversarial examples by perturbing an input slightly to induce incorrect behavior, which is checked for manually. However, these black-box approaches are completely unguided in terms of the absence of a coverage criterion and overlook the internal logic of a DNN. In DeepXplore [7], the authors introduced the concept of neuron coverage as a coverage metric for testing DNNs. They also proposed using multiple implementations for the same task as an oracle to avoid manual labeling effort. Further, DeepCover [8] proposes several criteria for testing DNNs, inspired by modified code/decision coverage for software testing. Their coverage criteria take into account the condition-decision dependence between neurons of consecutive layers. Another recent approach, DeepMutation [4] is the first source-level mutation testing technique that proposes a set of model-level mutation testing operators that directly mutate on deep learning models without a training process. DeepCT [5] uses a combinatorial-testing inspired coverage criterion which guides an exhaustive search for test inputs that activate neurons in a layer-wise manner.

IV. Opportunities for Improvement

A. *Why do we need a better coverage criteria?*

Coverage criteria for traditional software programs, such as code coverage and branch coverage check that all parts of the logic in the program have been tested by at least one test input and all conditions have been tested to independently affect the entailing decisions. On similar lines, any coverage criterion for deep neural networks must be able to guarantee completeness, that is, it must be able to ensure that all parts of the internal decision-making structure of the DNN have been exercised by at least one test input.

A typical feed-forward deep neural network contains multiple nonlinear processing layers with each hidden layer using the output of the previous hidden layer as its input. Each layer consists of multiple *neurons*. A *neuron* is a computing unit, loosely patterned on the neurons in the human brain, which fires/activates when it receives sufficient stimuli or input. Mathematically, if L_{k-1} and L_k denote two consecutive layers of this DNN (Figure 1):

$$n_{i,k} = \phi_k\Big(\delta_{i,k} + \sum_{1\leq j\leq N_{k-1}} (w_{j,i}\cdot n_{j,k-1})\Big) \quad (1)$$

where:

- $n_{i,k}$ denotes the value of the ith neuron of kth layer,
- ϕ_k denotes the activation function of the kth layer,
- $\delta_{i,k}$ denotes the bias for node $n_{i,k}$,
- N_{k-1} denotes the number of nodes/neurons of layer L_{k-1}, and
- $w_{j,i}$ denotes the weight of the connection between the jth neuron of layer L_{k-1} and ith neuron of layer L_k

Therefore, along with the value of each neuron having an independent effect on the activation of neurons in the next hidden layer, the combinations of values of neurons in the same layer also affect the value of neurons in the next layer. Any coverage criterion for deep neural networks must be able to capture both of these factors. Further, the coverage criterion should be scalable to larger-sized real-world DNNs and different network architectures. The coverage criteria proposed by previous works suffer from several limitations:

- Neuron coverage [7] measures the parts of the DNN's logic exercised by the test inputs based on the number of neurons activated by the input. However, it is not able to thoroughly account for all the possible behaviors that a DNN could exhibit. Experiments by [8] were able to prove that neuron coverage is fairly easy to achieve and 25 random test inputs are able to achieve close to 100% neuron coverage. Further, we observed that corner case behavior can be found beyond 100% neuron coverage. Our experiments[1] found that in certain cases of model architecture, for instance LeNet-1 used for MNIST handwritten digit classification, 100% neuron coverage can be obtained with two test inputs, because for most test inputs, the neurons are always fired/activated. Therefore, neuron coverage is a fairly coarse and insufficient criterion for coverage in DNNs.
- DeepCover's [8] coverage criteria take into consideration the condition-decision dependence in adjacent layers of a DNN. Apart from their method being tested on relatively small networks, it assumes the DNN to be a feedforward, fully-connected network and cannot generalize to architectures like RNNs, LSTMs, attention networks, etc. Such methods do not consider the context of a neuron in its own layer, and the combinations of neuron outputs in the same layer.
- DeepCT's [5] combinatorial testing inspired coverage criterion determines the fraction of logic exercised by a test input in terms of the fraction of neurons activated in each layer. It does not consider the inter-layer relationships within a DNN, and has not been verified to scale to real-world DNNs with different kinds of layers.

B. *Why do we need better test input generation?*

Generating or selecting test inputs in a guided manner usually has two major goals - maximizing the number of uncovered faults, and maximizing the coverage. [7] introduces a joint optimization based test input generation method, in which an existing test input is modified (using image manipulations) recursively until a test input causing differential behavior is found. [9] uses a similar greedy search technique in which random transformations are applied until an appropriate test input is found. Such test input generation methods suffer from some major drawbacks:

[1]All results for neuron coverage were obtained by running the DeepXplore code https://github.com/peikexin9/deepxplore/tree/master/MNIST for image manipulation=light and best-performing parameters: λ_1=1, λ_2=0.1, steps=10, grad_iterations=1000, threshold=0

- The iterative process of manipulating an existing test input until a test input that satisfies the criterion is found, has considerable time per execution.
- The number of test inputs that actually cause an increase in coverage and/or an increase in the number of uncovered corner case behaviors are fairly low in comparison to the sum of total number of tested and generated inputs.

C. *Why do we need a better oracle?*

Testing for the correctness of a DNN requires the presence of ground truth (oracle), that decides if the behavior is correct. The existing oracles for testing DNNs suffer from several limitations:

- The most straightforward way in data-driven schemes like DNNs is by collecting as much real-world data as possible and manually labeling it to check for correctness. However, such a process requires a lot of manual effort.
- In multiple DNN implementations [7] as an oracle, multiple implementations for the same task are compared, and differential behavior is labeled as a corner case behavior. However, we observed that this method erroneously classifies certain corner case inputs as correct behaviors because the labels predicted by all the implementations are similar and misclassifies several correct inputs as corner-case behaviors. Also, this method is only valid in applications that have several existing high-accuracy implementations. Often, DNNs may be deployed in tasks that do not have many existing implementations and/or implementations may be crafted by the same set of experts that are bound to have used the same methods or made the same errors.

V. Preliminary Approach and Results

In this paper, we focus only on proposing a finer coverage criterion. An ideal coverage criterion for deep neural networks must be able to guarantee completeness, i.e., all parts of the internal decision logic of the DNN have been tested by at least one input. Recall that the value of a neuron in a particular layer in a DNN is computed as a nonlinear function of the weighted sum of neurons in the previous layer, as shown in Equation 1. On these lines, we propose a coverage criterion that incorporates both factors- the conditional effect of each neuron on the value of neurons in the next layer and the combinations of values of neurons in a layer [3].

For two consecutive layers, L_{k-1} and L_k in a given (feed forward) deep neural network, let the neurons in these layers be denoted by $\{n_{1,k-1}, n_{2,k-1}, ..., n_{N_{k-1},k-1}\}$ and $\{n_{1,k}, n_{2,k}, ..., n_{N_k,k}\}$ respectively, where N_k denotes the total number of neurons in L_k. For any test input t, a neuron n is said to be activated if its value is greater than a certain threshold, for example, 0. Formally, if $\phi(t, n)$ denotes the activation of neuron n when the input to the deep neural network is t, then if $\phi(t, n) >0$ (or any other threshold value, depending on activation function) then the neuron is said to be activated or *fired*. Therefore, for a given neuron n and test input t, the condition $\phi(t, n) > 0$ can have two values, true or false, depending on whether the neuron is activated or not.

Based on these definitions, our coverage criterion is defined as the 2-way coverage [3] for every such triplet in the DNN: $(n_{i,k-1},\ n_{j,k-1},\ n_{q,k})$. Formally, for a given test set for *n* variables, simple *t-way* combination coverage is the proportion of *t-way* combinations of *n* variables for which all variable-values configurations are fully covered. By ensuring 2-way coverage on three such distinct neurons, we are able to cover, (1) the independent effect a condition (activation of neuron in L_{k-1}) has on an outcome (value of neuron in the next layer, L_k), (2) the failures that may arise because of the 'interaction' or activation values of neurons in the same layer L_{k-1}. While the first coverage is more inspired by MC/DC [2] and other traditional software-coverage criteria, the second coverage is inspired by combinatorial testing [3]. This kind of testing is based on the fact that not every parameter contributes to every failure, and empirical data suggest that *nearly all failures are caused by interactions between relatively fewer parameters*. This finding has important implications for testing because it suggests that testing combinations of (fewer) parameters can provide highly effective fault detection. In our scenario, since values of multiple (but not always all) neurons in the previous layer contribute towards the values of neurons in the next layer, such a method is able to test for multiple values that a condition (weighted sum in Equation 1) can take, which is also one of the requirements for traditional software coverage criteria such as MC/DC [2].

For preliminary results, we approach guided test input generation via joint optimization [7]. Any triplet not having achieved 100% coverage is randomly chosen to determine which combination(s) of activation values has not been covered. Consider, for example, the DNN instance where $\mathrm{n_{i,k\text{-}1}}$ is fired but $\mathrm{n_{j,k\text{-}1}}$ is not activated. The decision neuron $\mathrm{n_{q,k}}$ is fired. The objective becomes

$$F_{n,t} = f_{n_{i,k-1}}(t) + f_{n_{j,k-1}}(t) + f_{n_{q,k}}(t), \qquad (2)$$

where

- $f_{n_{i,k-1}}(t) = \phi(t, n_{i,k-1})$ needs to be maximized such that $\phi(t, n_{i,k-1}) > 0$ (or the decided threshold),
- $f_{n_{j,k-1}}(t) = \phi(t, n_{j,k-1})$ needs to be minimized such that $\phi(t, n_{j,k-1}) = 0$, and
- $f_{n_{q,k}}(t) = \phi(t, n_{q,k})$ needs to be maximized such that $\phi(t, n_{q,1}) > 0$.

The objective function to maximize coverage therefore becomes,

$$F_{n,t} = \phi(t, n_{i,k-1}) - \phi(t, n_{j,k-1}) + \phi(t, n_{q,k}). \qquad (3)$$

Because the individual terms in $F_{n,t}$ are activation values of certain neurons in certain layers and $\phi(t, n)$ for any n is a sequence of stacked functions, the gradient $\frac{\partial F_n(t)}{\partial t}$ can be calculated using the chain rule in calculus, i.e., by computing layer-wise derivatives backwards from the layer containing neuron n until reaching the input layer which takes input t [7].Hence, the input t can be manipulated in steps to maximize $F_{n,t}$. The oracle we use is the same as [7], so the second objective is to generate differential behavior causing inputs.

Metric	Result
Coverage for 10 random inputs	8.9%
Guided coverage for 550 test inputs	31%
Number of corner case behaviors found for 550 inputs tested	483
Adversarial Ratio	87.8%

TABLE I: Evaluation of coverage metric on LeNet architectures for MNIST dataset. All results are an average over LeNet-1, LeNet-4, LeNet-5.

We evaluated our coverage metric on three DNNs that classify the MNIST dataset of handwritten digits: LeNet-1, LeNet-4 and LeNet-5. Since the primary goal of our work is to introduce and test a more fine-grained coverage metric, our test input generation method and oracle share the limitations mentioned in section IV. The metrics used for determining the validity of our coverage criterion were:

- The coverage obtained on ten random test inputs, and
- The ratio of number of corner cases found to the number of total test inputs.

Ideally, the coverage for ten random test inputs (not generated using a guided method) must be low, i.e., the criterion must be difficult to achieve for random inputs, and the adversarial ratio must be high. We currently use multiple implementations [7] as an oracle, introduced in IV, and only one image manipulation, brightness. The results are summarized in Table I[2]. We then compared our proposed coverage criterion with existing coverage metrics. We found that for the same dataset and DNNs, the average neuron coverage [7] for ten random test inputs over the three DNNs is 30.5% (threshold used is 0), as opposed to 8.9% for our coverage criterion. Further, for LeNet-1, 100% neuron coverage can be achieved with just two corner-case inputs and a lot of corner-case inputs can be found beyond achieving 100% neuron coverage. On the other hand, LeNet-1 achieves close to 11.6% coverage for our criterion over 550 test inputs. This is because the most common activation pattern in the DNN for the given test inputs is all neurons being fired/activated, and hence 2-way coverage is difficult to achieve. The average neuron coverage across all three DNNs using guided test input generation is 98.5% for 550 test inputs, but is 31% for our proposed coverage criterion using the same test input generation method. The maximum adversarial ratio obtained using DeepCover [8] for DNNs of similar size is 11%. Similarly, DeepCT [5] achieves less than 10% adversarial ratio for a DNN of similar size, for 10,000 test inputs. These results confirm that for testing DNNs, it is important to have a more fine-grained coverage metric that not only incorporates inter-layer relationships, but also the relative activations of neurons in the same layer. While the large number of triplets may seem like a computational bottleneck, the average time taken to update coverage for LeNet-5 with the most number of triplets (651720) is 2.08 seconds.

[2]Our implementation involves trying to optimize for image manipulations to generate differentially behaving, more coverage test inputs from *all* inputs, whether or not they cause differential behavior when not manipulated at all.

VI. Conclusion

The absence of a transparent decision logic makes it impossible to apply traditional software testing methods to DNNs. This paper examines existing testing methods for deep neural networks and recognizes several limitations such as coarse coverage criteria, open ended processes, unreliable oracles, inefficient test input generation methods, inability to scale to larger DNNs and different network architectures, etc. Further, we propose a fine-grained coverage criterion for feed forward DNNs that takes into account the condition-decision relationships between adjacent layers and the combinations of values of neurons in the same layer. A set of ten random test inputs could only achieve 8.9% of our coverage criterion. Further, when coupled with gradient-based search techniques and multiple implementations oracle, it is able to achieve an average 87.8% adversarial ratio over three models. The ability to test the internal logic of a DNN to a greater extent makes its performance better than existing methods. The scalability of the coverage method to larger-sized real-world DNNs and its adaptation to different network architectures is yet to be tested.

VII. Acknowledgements

This material is based upon work supported in part by the National Science Foundation under Grant No. CNS: 1650512, conducted in the NSF UICRC Center of Visual and Decision Dynamics. This research was also supported by the Northrop Grumman Mission Systems University Research Program.

References

[1] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to End Learning for Self-Driving Cars. 2016.

[2] Kelly J Hayhurst, John J Chilenski, and Leanna K Rierson. A Practical Tutorial Decision Coverage on Modified Condition. Technical report, 2001.

[3] D Richard Kuhn, Raghu N Kacker, and Yu Lei. ”Combinatorial Testing”. Technical report.

[4] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. DeepMutation: Mutation Testing of Deep Learning Systems. 2018.

[5] Lei Ma, Fuyuan Zhang, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, and Yadong Wang. Combinatorial Testing for Deep Learning Systems. 2018.

[6] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple Black-Box Adversarial Perturbations for Deep Networks. 2016.

[7] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. 2017.

[8] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing Deep Neural Networks. 2018.

[9] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars. 2017.

[10] Y. LeCun. Lenet-5, convolutional neural networks.