

Test Metrics for Recurrent Neural Networks

Wei Huang¹, Youcheng Sun², James Sharp³, Xiaowei Huang¹

Abstract—Recurrent neural networks (RNNs) have been applied to a broad range of application areas such as natural language processing, drug discovery, and video recognition. This paper develops a coverage-guided test framework, including three test metrics and a mutation-based test case generation method, for the validation of a major class of RNNs, i.e., long short-term memory networks (LSTMs). The test metrics are designed with respect to the internal structures of the LSTM layers, to quantify the information of the forget gate, the one-step information change of an aggregate hidden state, and the multi-step information evolution of positive and negative aggregate hidden state, respectively. We apply the test framework to several typical LSTM applications, including a network trained on IMDB movie reviews for sentiment analysis, a network trained on MNIST dataset for image classification, and a network trained on a lipophilicity dataset for scientific machine learning. Our experimental results show that the coverage-guided testing can be used to not only extensively exploit the behaviour of the LSTM layer in order to discover the safety loopholes (such as adversarial examples) but also help understand the internal mechanism of how the LSTM layer processes data.

I. INTRODUCTION

Recurrent neural networks (RNNs) have been widely used in many application areas such as automated language translation [45], robotic control [42], drug discovery [36], automatic speech recognition [19], time series prediction [33], etc. Most efforts for developing RNNs have been spent on improving empirical accuracy and reducing empirical generalisation error, and less work has been done towards their verification and validation. Verification and validation (V&V) are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it fulfills its intended purpose [2]. Unlike software systems which are programmed based on their specifications, a learning system is obtained by learning from a set of data samples. That is, the specification of a learning system is less explicit. While research has started to discuss how to establish specifications for learning systems [5], this paper is moving towards developing techniques to validate RNNs against their specifications. More specifically, we focus on a major class of RNNs, i.e., long short-term memory networks (LSTMs) and an important specification.

The specification to be considered is the robustness, which requires that a prediction made by a learning system is invariant with respect to small perturbations on the input. It has been shown that robustness may be at odds with accuracy [40], which suggests that, in addition to improving the empirical accuracy, efforts are needed to check, and improve, the robustness of learning systems. Since the discovery of adversarial examples in deep feedforward neural networks (FNNs), particularly convolutional neural networks (CNNs) for image classifica-

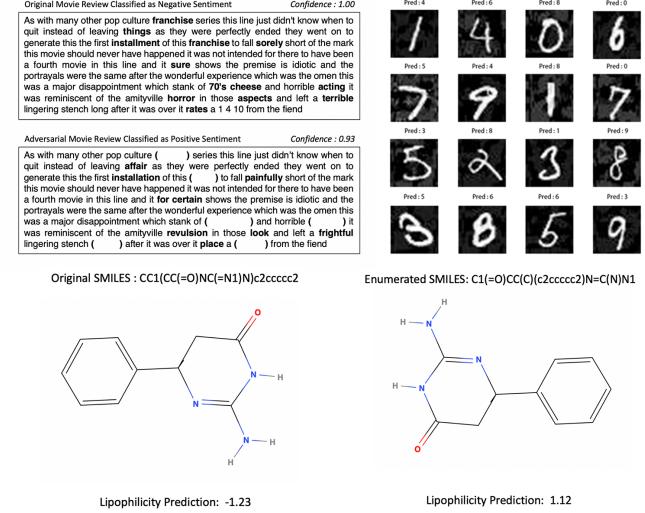


Fig. 1. Adversarial examples for LSTM models trained on MNIST handwritten digits dataset for image classification, an IMDB reviews dataset for sentiment analysis, and a SMILES strings dataset for lipophilicity prediction, respectively.

tion [39], the robustness of neural networks has been intensively studied from several aspects including e.g., attack methods [7], defence methods [28], verification methods [24], [21], [16], and testing methods [32], [38], [26], etc. However, the research on the verification and validation methods for RNNs are sparse, if there is any, due to the challenging facts that (1) verification and testing methods are usually white-box, which requires the access to internal structures of a network, (2) the internal structures of RNNs are much more complicated than CNNs, with some continuous structures such as sigmoid and tanh cannot be directly handled with existing verification methods such as SMT-based methods [24] and its iterative nature easily leads to the scalability problem for search-based methods [21] and abstract interpretation based methods [16], and (3) existing deep learning platforms such as Tensorflow do not support a direct, easy access to the internal structures of LSTM layers. Nevertheless, the verification and validation of RNNs are needed, see Figure 1 for a few typical adversarial examples (the faults with respect to the robustness) for LSTM networks used in our experiments.

Our approach is based on the coverage-guided testing [47], which has been shown successful in software fault detection. Coverage-guided testing has been extended to work with FNNs in recent work such as [32], [44], [26], [37], [38], [25], where a collection of test metrics and test case generation algorithms are proposed. These metrics are based on the structural in-

formation of the FNNs, such as the neurons [32], [26], the relation between neurons in neighboring layers [37], [38], etc. By generating a set of test conditions and a set of test cases, a coverage-guided testing exploits the behaviour of network by claiming that a certain percentage of test conditions are asserted or satisfied by the test cases. When working with RNNs whose internal structures are much more involved, new test metrics and new test case generation methods are needed to take into account the additional structures and complexity.

We develop three test metrics and a generic mutation-based test case generation method. The test metrics are designed to exploit different functional components of the LSTM networks, by considering both the one-step information change and the multi-step information evolution, and both the gate vectors and the hidden state vectors. Gate and hidden state are internal structural components of an LSTM cell. Specifically, we quantify the filtering ability of forget gates with gate coverage (GC), the one-step change of aggregated hidden states with cell coverage (CC), and the multi-step evolution of positive or negative hidden states with sequence coverage (SC). For test case generation, we iteratively apply a set of mutation operations on seed inputs. When a gradient of the network loss function with respect to the input is obtainable (for continuous inputs such as images), we design a set of mutation operations based on the gradient direction. On the other hand, when a gradient is not obtainable (for discrete inputs such as words), we use user-specified mutation operations.

In [], the effectiveness of structural coverage guided testing is questioned. The authors mentioned a few aspects, including the structural coverage can be too loose, the test case generation is dependent on the adversarial attacking algorithm, and there is a lack of correlation between misclassified inputs in a test set and its structural coverage.

We develop a tool **testRNN**¹ and conduct an extensive set of experiments on three LSTM networks, trained on an IMDB review dataset, the MNIST image dataset, and a lipophilicity (a physicochemical property of drugs) dataset, respectively. Two of them have a single LSTM layer and the other has two connected LSTM layers. The experimental results show that, (1) our test framework is effective, in terms of its efficiency in generating test cases, its (non-trivial) possibility in achieving high coverage, and its ability in discovering faults, (2) the test metrics are complementary to each other in guiding the generation of test cases, i.e., the test cases generated by one test metric cannot be applied to achieve high coverage of the other metrics, and (3) the generated test cases can be utilised to understand the working principle of the LSTM networks.

II. PRELIMINARIES

Feedforward neural networks (FNNs) can be represented as a function $\phi : X \rightarrow Y$, mapping from input domain X to output domain Y , and is usually used to perform predictions based on an input $x \in X$, or recognise patterns in x . For a sequence of inputs x_1, \dots, x_n , ϕ will handle them individually

¹ Available via <https://github.com/TrustAI/testRNN>

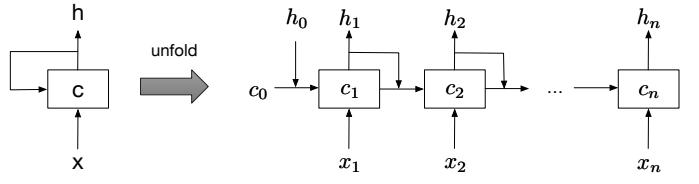


Fig. 2. A simple recurrent layer

without considering results from previous predictions, that is, the result of $\phi(x_i)$ is independent of the results of $\phi(x_j)$ when $j \neq i$.

By contrast, a recurrent neural network (RNN) is used to work with sequential data. Usually, an RNN contains at least one recurrent layer. Figure 2 illustrates how a sequential input is handled by a recurrent layer by unfolding. A recurrent layer can be represented as a function $\psi : X' \times C \times Y' \rightarrow C \times Y'$ such that $\psi(x_t, c_{t-1}, h_{t-1}) = (c_t, h_t)$ for $t = 1 \dots n$, where c_t is the cell state and acts as the intermediate memory, and t denotes the t -th time step. Intuitively, the recurrent layer takes as inputs the current time input x_t , the previous time memory state c_{t-1} and the previous time output h_{t-1} , updates the memory state into c_t , and returns h_t as the current time output. Initially, we let c_0 and h_0 be 0-valued vectors. For a (finite) sequence of inputs x_1, \dots, x_n , this function ψ is applied recursively on them. For example, the popular long short-term memory (LSTM) layer can be represented with the following equations for time t :

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ c_t &= f_t * c_{t-1} + i_t * \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \\ o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= o_t * \tanh(c_t) \end{aligned} \quad (1)$$

where σ is the sigmoid function such that $\sigma(x) \in [0, 1]$ for any $x \in \mathbb{R}$, \tanh is the hyperbolic tangent function such that $\tanh(x) \in [-1, 1]$ for any $x \in \mathbb{R}$, W_f, W_i, W_c, W_o are weight matrices, b_f, b_i, b_c, b_o are bias vectors, f_t, i_t, o_t are internal gate variables, h_t is the hidden state variable (utilising o_t), and c_t is the cell state variable. For the connection with successive layers, we only take the last output h_n as the output. For simplicity, when working with finite sequential data, we can also define a recurrent layer as $\psi : (X')^n \rightarrow Y'$, which takes, as input, a sequential data of length n and returns the last output h_n . Normally, the recurrent layer is connected to non-RNN layers such as fully connected layers so that the output h_n is processed further. We let the remaining layer be a function $\phi_2 : Y' \rightarrow Y$. Moreover, there can be feedforward layers connecting to the RNN layer, and we let it be a function $\phi_1 : X \rightarrow X'$. Then given a sequential input x_1, \dots, x_n , the RNN is a function φ such that

$$\varphi(x_1 \dots x_n) = \phi_2 \cdot \psi(\prod_{i=1}^n \phi_1(x_i)) \quad (2)$$

In general, an RNN is a function $\varphi : X^n \rightarrow Y$ which takes, as input, a sequence x_1, \dots, x_n . Inputs x_i are processed in parallel with the function ϕ_1 , with the results being processed within the RNN layer ψ . The result from the RNN layer is then processed further with function ϕ_2 to provide the final output. While the function φ is for networks with a single RNN layer, it can be easily extended to networks with multiple RNN layers.

Deep (recurrent) neural networks, φ , consist of multiple layers, and we write L_k for the k -th layer. We write $N_k = \{n_{k,i} \mid 1 \leq i \leq s_k\}$ for the set of variables (or neurons) of layer L_k , where $n_{k,i}$ is the i -th variable among the s_k number of variables at layer L_k .

III. INTERNAL INFORMATION OF LSTM CELLS

In this part, we first formulate two types of internal information of LSTMs that will be used later to define our coverage criteria. For an LSTM network, there are a sequence of inputs $x = \{x_t\}_{t=1}^n$, gate values $f = \{f_t\}_{t=0}^n$, $i = \{i_t\}_{t=0}^n$, $o = \{o_t\}_{t=0}^n$, and hidden states $h = \{h_t\}_{t=0}^n$, where each element x_t, f_t, i_t, o_t, h_t is a vector of real numbers. We use variable v to range over $\{x, f, i, o, h\}$, use v_t to denote the value of v at time t , and write $v_t(k)$ to denote the k -th component of v_t . Moreover, for either v or v_t where $v \in \{f, i, o, h\}$, we may use a subscript x , written as e.g., $v_x, v_{x,t}$, etc., when it is needed to refer to its corresponding input x .

In the following, we consider abstract information from the internal structure, i.e., hidden state vectors and gate vectors, of an LSTM cell, and present their intuitive meanings. These information will be used to design test metrics in Section IV.

a) Aggregate information of hidden states: According to Equation (1), the component value of h_t varies between -1 and 1. It is therefore reasonable to divide the possible information of a component $h_t(k)$ into positive (> 0) and negative (< 0). For h_t , we take the aggregate information [29] $\xi_t = (\xi_t^+, \xi_t^-)$, by summing up positive component values and negative ones, respectively, such that

$$\begin{aligned}\xi_t^+ &= \sum \{h_t(i) \mid i \in \{1, \dots, |h_t|\}, h_t(i) > 0\} \\ \xi_t^- &= \sum \{h_t(i) \mid i \in \{1, \dots, |h_t|\}, h_t(i) < 0\}\end{aligned}\quad (3)$$

Intuitively, ξ_t^+ represents the extent to which the hidden state h_t contains positive information and ξ_t^- represents the extent to which the hidden state h_t contains negative information. In the following, we show how ξ_t can be utilised to provide intuition into the processing of inputs by LSTM layer. Basically, we consider both a one-step change and a multi-step change on information based on ξ_1, \dots, ξ_n .

Consider a sentiment analysis example, where an LSTM network is trained to classify the sentiment (positive or negative) of movie reviews. As shown in Figure 3, assume that the input is e.g., “horrible movie and really bad watching experience”, “I really liked the movie and had fun”, etc., we consider the following quantity

$$\Delta\xi_t = |\xi_t^+ - \xi_{t-1}^+| + |\xi_t^- - \xi_{t-1}^-| \quad (4)$$

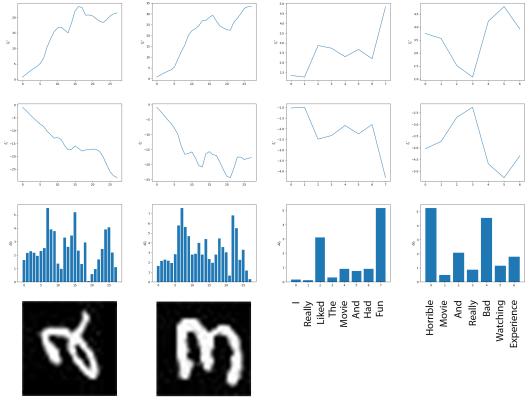


Fig. 3. Four examples to show how positive and negative elements of hidden state vectors represent the information in MNIST and IMDB models. The x-axis includes the inputs (bottom row) and the y-axis includes ξ_t^+ (top row), ξ_t^- (second row) and $\Delta\xi_t$ (third row) values.

which compares the information of the current step ξ_t with its previous step ξ_{t-1} . Intuitively, $\Delta\xi_t$ is the abstract representation of information updated between hidden states. From the bar charts in the third row of Figure 3, we can see that sensitive words such as “like”, “horrible”, “fun” trigger greater $\Delta\xi_t$ values than non-sensitive words such as “movie”, “really”, and “had”. Also, for MNIST images where the t -th column is processed as the cell input x_t , more informative columns such as the 9th and 10th columns of both the digit “2” and “3” trigger greater $\Delta\xi_t$ values than the first and last columns.

We can track the change of ξ_t^+ and ξ_t^- for a certain time span. Figure 3 presents two curves for each input. The one in the top row is for ξ_t^+ and in the second row is for ξ_t^- .

b) Abstract information of gates: We now consider information represented in the gates f , i , and o . In the LSTM, these gates have their intuitive meanings. For example, the forget gate f is to control the portion of information that should pass through the gate, the input gate i determines how much information will be added to the cell state, and the output gate o determines which part of the cell state to output.

Definition 1: Let $f_{x,t}$ be the value of the gate f at time t when the input is x . We use remember rate $R_t(f, x)$ to denote the portion of information passing through the gate f at time t that can be remembered, given the input is x . Formally, we have

$$R_t(f, x) = \frac{1}{|f_{x,t}|} \sum_{i=1}^{|f_{x,t}|} f_{x,t}(i). \quad (5)$$

It is easy to see that $R_t(f, x)$ is within the range $[0, 1]$, since all components in $f_{x,t}$ have their values bounded in $[0, 1]$.

IV. TEST METRICS FOR LSTM

Existing test coverage metrics work for neural networks [32], [26], [37], [38], [44], [25] focuses on feedforward structures such as convolutional layer, fully connected layer, maxpooling layer, etc. In this paper, we propose a set of structural coverage metrics to work with LSTM layers and

sequential inputs, by considering the internal information of the cells as discussed in Section III.

Let \mathcal{R} be a set of sets of test conditions, \mathcal{A} a set of assert methods, and \mathcal{T} a set of test suites. A test coverage metric on a network \mathcal{N} is a function $M_{\mathcal{N}} : \mathcal{R} \times \mathcal{A} \times \mathcal{T} \rightarrow [0, 1]$. Intuitively, $M_{\mathcal{N}}(\mathcal{R}, \mathcal{A}, \mathcal{T})$ quantifies the degree of adequacy to which a DNN \mathcal{N} is tested by a test suite \mathcal{T} with respect to a set \mathcal{R} of test conditions and an assert method \mathcal{A} . Usually, the greater the number $M_{\mathcal{N}}(\mathcal{R}, \mathcal{A}, \mathcal{T})$, the more adequate the testing. In this section, we will develop the set \mathcal{R} and for every $\mathcal{R} \in \mathcal{R}$ their corresponding assert method \mathcal{A} and metric $M_{\mathcal{N}}(\mathcal{R}, \mathcal{A}, \mathcal{T})$. A test suite $\mathcal{T} \in \mathcal{T}$ contains a set of test cases, and we will discuss in Section VI an algorithm we use in our experiments to generate test suites.

A. Test Conditions

Every test condition is some functionality or behaviour of the system we want to verify. In the following, we introduce methods to define test conditions for LSTM networks.

a) Neuron Coverage: As a simple baseline, we extend the idea of neuron coverage [32] to work with LSTM layers. Let \mathcal{R}_{NC} be the set of test conditions, each of which represents a hidden neuron under consideration. For LSTM layer, \mathcal{R}_{NC} includes the component neurons in the last output h_n , for an input x of length n . Note that, \mathcal{R}_{NC} does not include component neurons in any h_i for $i < n$, since they are intermediate results of LSTM and not obtainable by simply observing the output of an LSTM layer.

Definition 2: Assume that we are working with the k -th layer which is an LSTM layer. A neuron $n_{k,i}$ is asserted by a test case x , denoted as $\mathcal{A}_{NC}(n_{k,i}, x)$, if $h_{x,n}(i) > 0$. Recall that, we use $h_{x,n}(i)$ to denote the i -th component of the hidden state h_n at the last time step n when given the input x , and $h_{x,n}(i) \in [-1, 1]$.

As will be shown in the experiments in Section VIII, the 100% neuron coverage (the concept of coverage will be formally defined later in Section IV-B) can be easily achieved with a small number of test cases. This makes neuron coverage less powerful for testing, which aims to exploit the system with a non-trivial number of test cases in order to expose incorrect behaviour of a network. One reason for neuron coverage to be less strong is that, it treats neurons as independent units, without considering their collective behaviour. However, the key strength of neural networks is its capability of extracting features (which are usually represented with a set of neurons) and based on this, combining a set of features to conduct high-level tasks such as pattern recognition. It is not hard to see that a feature in itself can be harder to be asserted by a set of test cases than the neurons supporting it.

Instead of considering only the individual neurons, in the following we design several test metrics for LSTM layers that take into account the internal information of a vector of neurons. In particular, the aggregate information of hidden states and the abstract information of gates in Section III will be used.

b) Gate Coverage: The first set of new test conditions is designed to cover the behaviour of internal gates, i.e., f, i, o . In this paper, we use the forget gate f as an example. The behaviour of the forget gate at time t , i.e., f_t , affects not only the behaviour of the cell at time t locally but also the overall behaviour of the LSTM layer. We let

$$\mathcal{R}_{GC} = \{(f_t, \alpha_f) \mid t \in \{1..n\}\}$$

be the set of test conditions, such that f_t is the forget gate value at time t and $\alpha_f \in [0, 1]$ is a threshold value .

Definition 3: A test condition (f_t, α_f) is asserted by a test case x , denoted as $\mathcal{A}_{GC}((f_t, \alpha_f), x)$, if $R_t(f, x) > \alpha_f$, i.e., the forget rate at time t is greater than the designated threshold α_f .

An important benefit for studying forget gate f_t is on understanding the cell state c_t . According to Equation (1), f_t is used to directly control the information flowing from c_{t-1} to c_t . However, because c_t represents long-term memory, its component values are not bounded, which makes the design of test conditions harder to be fully automated, i.e., may have to be problem specific. The other benefit is that, gate activation statistics have been used to explain the internal mechanisms of LSTM [23], and as will be discussed in the experiments, the study of forget gates enables our understanding of the learning process through a memorisation curve.

c) Cell Coverage: Cell is the basic working unit in LSTM to deal with the information flow. According to Equation (1), in addition to the gates (forget gate f_t and internal gate i_t), the hidden state h_{t-1} is also a factor determining the value of c_t . The first sets of test conditions on hidden state h_t is based on quantifying the one-step change of the hidden output Δh_t , which is measured by the hidden aggregate information change $\Delta \xi_t$. Therefore, we let

$$\mathcal{R}_{CC} = \{(\Delta \xi_t, \alpha_h) \mid t \in \{1..n\}\}$$

be the set of test conditions, such that $\Delta \xi_t$ is the hidden aggregate information change at time t and $\alpha_h > 0$ is a threshold value.

Definition 4: A test condition $(\Delta \xi_t, \alpha_h)$ is asserted by a test case x , denoted as $\mathcal{A}_{CC}((\Delta \xi_t, \alpha_h), x)$, if $\Delta \xi_{x,t} > \alpha_h$. Recall that, we use $\Delta \xi_{x,t}$ to refer to the value $\Delta \xi_t$ when the input is x .

d) Sequence Coverage: As explained earlier, we are also interested in checking, for an LSTM model, how information updates in the time series. This complex task is essentially a time series classification problem, and we can employ symbolic representation method to provide a specific metric to divide the hidden sequence patterns into several classes. Our interest localizes on the sequence of positive hidden information ξ^+ and negative information ξ^- .

Definition 5 (Symbolic Representation of Time Series): Let $D(\xi^+)$ be the (possibly unbounded) range of values of the quantity ξ_t^+ . Given a finite set Γ of symbols, we split $D(\xi^+)$ into a set of sub-ranges

$$\Phi_{\Gamma}(D(\xi^+)) = \{D_i(\xi^+) \mid i \in \Gamma\},$$

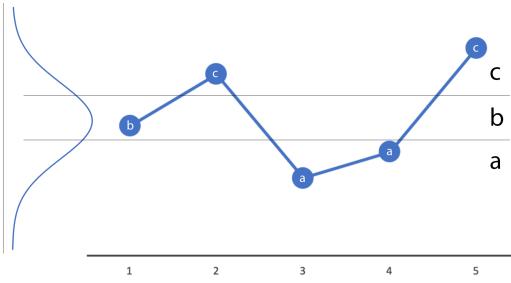


Fig. 4. A time series is transformed into symbolic representation with three symbols $\Gamma = \{a, b, c\}$

such that $D(\xi^+) = \bigcup_{i \in \Gamma} D_i(\xi^+)$ and for all $i, j \in \Gamma$, $i \neq j$ implies that $D_i(\xi^+) \cap D_j(\xi^+) = \emptyset$. Based on $\Phi_\Gamma(D(\xi^+))$, for any value of ξ_t^+ , we associate it with a symbol $i \in \Gamma$ whenever $\xi_t^+ \in D_i(\xi^+)$. One step further, every sequence $\xi^+ = \xi_1^+ \xi_2^+ \dots \xi_n^+$ can be associated with a clause $i_1 \dots i_n$ such that $\xi_j^+ \in D_{i_j}(\xi^+)$ for all $j \in \{1..n\}$.

While Definition 5 is for ξ^+ , it can be extended to work with ξ^- . Intuitively, it discretises the domain of the continuous quantity ξ_t^+ into a set of finite sub-ranges $\Phi_\Gamma(D(\xi^+))$, associates each sub-range with a symbol in Γ , and then projects each component of a time series into a symbol by matching its value with a sub-range. By this process, a time series is transformed into a sequence of symbols, or a clause.

Practically, the symbolic representations of time series are obtained by the following steps, as illustrated intuitively in Figure 4. First, a large amount of sequence data are sampled to have their distribution. Second, a set of breakpoints are determined by dividing the area under the curve into a set of sub-areas of equal size volume. Every sub-area is associated with a symbol. This step is to ensure that the appearance of symbols has equal probability. Finally, a clause is generated for each time series by replacing the vertices with symbols corresponding to their respective sub-areas.

After getting the symbolic representation for time sequences, we can define test conditions. Given $\Phi_\Gamma(D(\xi^+))$ and a range $[a, b] \subseteq [1..n]$, we let

$$\mathcal{R}_{SC}^{[a,b]} = \{\perp \dots \perp i_a i_{a+1} \dots i_b \perp \dots \perp \mid i_j \in \Gamma, j \in \{a..b\}\} \quad (6)$$

be a set of test conditions, where \perp is a special symbol representing that the place is not within the range $[a, b]$. Given a sequence $s \in \mathcal{R}_{SC}^{[a,b]}$, we write $s(i)$ for $i \in \{1..n\}$ to denote its i -th component.

Definition 6: Given a test case x and a test condition $s \in \mathcal{R}_{SC}^{[a,b]}$, we say that s is asserted by x with positive information, denoted as $\mathcal{A}_{PSC}(s, x)$, if for all $j \in \{1..n\}$, we have that $s(j) \neq \perp$ implies that $\xi_j^+(x) \in D_{s(j)}(\xi^+)$. Moreover, we say that s is asserted by x with negative information, denoted as $\mathcal{A}_{NSC}(s, x)$, if for all $j \in \{1..n\}$, we have that $s(j) \neq \perp$ implies that $\xi_j^-(x) \in D_{s(j)}(\xi^-)$.

Intuitively, sequence coverage is designed to test all possible time series of either the positive information ξ^+ or the negative

information ξ^- .

B. Test Metrics

Let $\mathcal{C} = \{NC, GC, CC, PSC, NSC\}$, $\mathcal{R} = \{\mathcal{R}_C \mid C \in \mathcal{C}\}$ and $\mathcal{A} = \{\mathcal{A}_C \mid C \in \mathcal{C}\}$, as defined in Section IV-A. Based on them, we define a generic metric as follows.

Definition 7: Let $C \in \mathcal{C}$ and \mathcal{T} a set of test cases. The test coverage metric $M_N(\mathcal{R}_C, \mathcal{A}_C, \mathcal{T})$ for a network N is defined as the percentage of the test conditions in \mathcal{R}_C asserted by the test suite \mathcal{T} with method \mathcal{A}_C . Formally,

$$M_N(\mathcal{R}_C, \mathcal{A}_C, \mathcal{T}) = \frac{|\{\exists x \in \mathcal{T} : \mathcal{A}_C(r, x) \mid r \in \mathcal{R}_C\}|}{|\mathcal{R}_C|} \quad (7)$$

V. TEST ORACLE

Test oracle is usually employed to determine if a test passes or fails. Test oracle automation is important to remove a bottleneck that inhibits greater overall test automation [6]. For our testing of LSTM networks, we employ a constraint (or more broadly, a program) to determine whether a generated test case passes the oracle or not. A test case does not pass the test oracle suggests a potential safety loophole. It is usually for the designer (or developer) of the LSTM network to determine which constraint should be taken as the oracle, since the latter relates to the correctness specification of the network under design. In this paper, as an example to exhibit our test framework (i.e., test metrics and test case generation algorithm), we take the currently intensively-studied adversarial example problem [39] to design our test oracle, and remark that the test oracle in general is orthogonal to the test framework we developed in this paper.

Practically, to work with adversarial examples, we define a set of norm-balls, each of which is centered around a data sample with known label. The radius α_{oracle} of norm-balls is given to intuitively mean that a human cannot differentiate between inputs within a norm ball. In this paper, we consider Euclidean distance $\|\cdot\|_2$. We say that a test case x' does not pass the oracle if and only if (1) x' is within the norm-ball of some training sample x , i.e., $\|x - x'\|_2 \leq \alpha_{oracle}$, and (2) x' has a different classification from x , i.e., $\varphi(x) \neq \varphi(x')$.

VI. TEST CASE GENERATION

We present a generic test case generation algorithm used in our experiments. To avoid “gaming against criteria”, our test case generation does not use test metrics as targets, which is recommended by Chilensky and Miller in their seminal work [11]. Given a LSTM network and a specified test metric, a test suite is generated: the coverage result and the number of adversarial examples out of the test suite are used to evaluate the robustness of the LSTM network under test. The effectiveness of our test generation algorithm is later justified by experiments in Section VIII.

The test generation in Algorithm 1 starts from a set of seed inputs \mathcal{T}_0 , and the function m is a pre-defined mutation function. Initially, the test suite \mathcal{T} contains only inputs from \mathcal{T}_0 (Line 1). A dictionary $orig$ is defined (Line 2) to map every

test case/mutant generated to its original input in \mathcal{T}_0 , and every seed input from \mathcal{T}_0 is mapped to itself (Line 3).

As long as the coverage condition, such as 100% neuron coverage, is not satisfied (Line 4), the test generation loop iterates. At each iteration, a test case x is sampled from the present \mathcal{T} (Line 5) and it is mutated using m to have a new input x' (Line 6). To guarantee the quality of test cases generated, x' is only added into the test suite \mathcal{T} if it falls within the norm ball with respect to its original seed input (Lines 7 – 9). Finally, the generated test suite \mathcal{T} is returned (Line 10).

Algorithm 1: A Generic Test Case Generation Algorithm

```

Input:  $\mathcal{N}$ : DNN to be tested
 $\mathcal{T}_0$ : a set of seed inputs
 $m$ : a mutation function
Output:  $\mathcal{T}$ : a set of test cases
1  $\mathcal{T} = \mathcal{T}_0$ 
2  $orig \leftarrow dict()$ 
3  $orig[x] \leftarrow x$  for all  $x \in \mathcal{T}_0$ 
4 while coverage condition is not satisfied do
5    $x \leftarrow$  to sample an element in  $\mathcal{T}$ 
6    $x' \leftarrow m(x)$ 
7   if  $\|orig(x) - x'\|_2 \leq$  pre-defined distance  $b$  then
8      $\mathcal{T} \leftarrow \mathcal{T} \cup \{x'\}$ 
9      $orig[x'] \leftarrow orig[x]$ 
10 return  $\mathcal{T}$ 
```

The test case generation in Algorithm 1 is indeed very generic and the core part is defining the mutation function m . In the following, we introduce the mutation operations used in this paper.

a) *Gradient-Based Mutation*: For the models where the gradient of the network loss function with respect to the input is obtainable (for continuous inputs such as images), we can utilise Algorithm 1 to search the input space along the gradient direction. Let $J(f, x, y)$ be a cost function over the model f , with input x , and output y , and $z \subseteq \bigcup_k N_k$ be a set of variables (hidden or not). We can define gradient as

$$grad(x) = \nabla_z J(f, x, y) \quad (8)$$

to denote the gradient of the loss $J(f, x, y)$ with respect to z . Normally, we take z to be N_k for some layer L_k . For example, $\nabla_{N_k} J(f, x, y)$ represents the usual gradient over the input dimensions. Finally, we let

$$\begin{aligned} m(x) &= f^t(x) \\ f^t(x) &= f^{t-1}(x) + \epsilon * sgn(grad(f^{t-1}(x))) \end{aligned}$$

where $f^0(x) = x$. That is, the gradient based mutation takes two parameters ϵ and t .

b) *Random Mutations*: For models where the gradient of the cost function is not obtainable (having discrete inputs such as words), we define a set \mathcal{M} of mutation functions that randomly mutate an input. At each step of the test generation, m is instantiated by one function in \mathcal{M} . Given different kinds

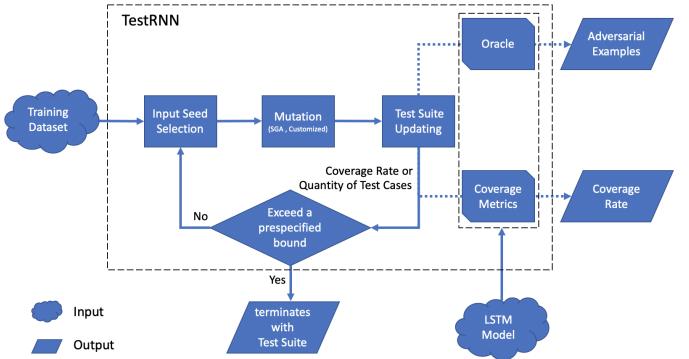


Fig. 5. Test Framework

of input data, the randomisation operations allowed may also differ. Our mutation operations used in the experiment are detailed in Section VIII-B.

VII. TEST FRAMEWORK

Figure 5 presents the overall architecture of our test framework for LSTMs. Given the LSTM network \mathcal{N} , a training dataset X , an oracle O (Section V), the test metric M_N (Section IV), and a mutation method m (Section VI), it first selects a set of seed inputs, and then the test case generation method in Algorithm 1 is applied to generate a test suite. The test generation terminates when certain level of coverage rate has been satisfied or certain number of test cases have been generated. We can then evaluate \mathcal{T} : (1) by considering oracle, we obtain the information about adversarial example; (2) by considering test metric, we obtain the coverage rate. Practically, the adversarial example can be used for data-augmentation. Given that this is a method well explored in other works such as [39], we did not pursue this direction in the paper. The coverage rate is used to determine if the test suite \mathcal{T} is adequately tested.

VIII. EXPERIMENTAL RESULTS

This section presents our experimental results on several typical LSTM networks. We evaluate the effectiveness of coverage-guided LSTM testing using adversarial examples as a proxy, we compare different metrics and show their complementary in guiding test case generation, and we utilise the generated test cases to access and understand the the LSTM internal mechanism. All the experiments are run on a CPU sever with Intel(R) Xeon(R) CPU E5-2603 v4 @ 1.70 GHz, 64 GB Memory.

A. LSTM Models

In the experiments, we consider the use of LSTM networks in different domains, including a network trained on MNIST dataset for image classification, a network trained on an IMDB review dataset for sentiment analysis, and a network trained on a Lipophilicity dataset for molecular machine learning.

a) *MNIST Handwritten Digits Analysis by LSTM*: The MNIST database, containing a set of 60,000 grey-scale images of size 28×28 , is used to train a HNN model with 4 layers. Figure 6 depicts its structure. The first two layers are LSTM layers, which are correspondingly connected and fed with rows of input images. That is, each input image is encoded as the row vector of shape $(28, 128)$ by the first LSTM layer, and then second layer will do further processing to output a image vector representing the whole image. Finally, two fully-connected layers with ReLU and SoftMax activation functions respectively, are used to process the extracted feature information to get the classification result. The model achieves 99.2% accuracy in training dataset (50,000 samples) and 98.7% accuracy in the default MNIST test dataset (10,000 samples).

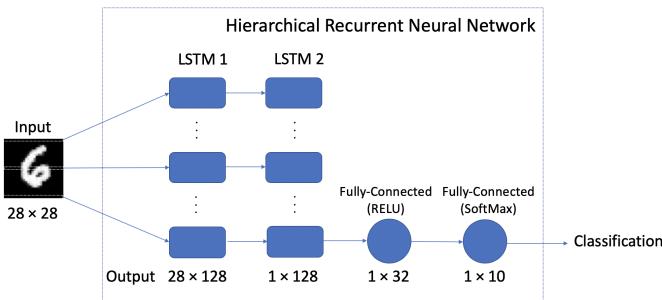


Fig. 6. An illustration of an LSTM network trained on MNIST database.

b) *Sentiment Analysis by LSTM*: The sentiment analysis network [1] has three layers, i.e., an embedding layer, an LSTM layer, and a fully-connected layer, with 213,301 trainable parameters. The embedding layer takes as input a vector of length 500 and outputs a 500×32 matrix, which is then fed into the LSTM layer. Subsequently, there is a fully-connected layer of 100 neurons.

c) *Lipophilicity Analysis by LSTM*: We trained an LSTM network from a Lipophilicity dataset downloaded from the MoleculeNet [46]. The model has four layers: an embedding layer, an LSTM layer, a dropout layer, and a fully connected layer. The input is a SMILES string (Figure 8) representing a molecular structure and the output is its prediction of Lipophilicity. A dictionary is used to map the symbols in the SMILES string to integers. We use the length of the longest SMILES in training dataset as the number of cells for the LSTM layer. Similar to text processing in the IMDB model, short SMILES inputs are padded with 0s to the left side. We use the root mean square error (RMSE) as the measurement of model accuracy. Our model is trained up to RMSE = 0.2371 in training dataset and RMSE = 0.6278 in test dataset, which are better than the traditional and convolutional methods used in [46].

B. Experimental Setting

a) *Symbolic Representation based on Empirical Distribution*: As discussed in Section IV, the symbolic representation

for a time series is obtained by sampling a number of data and splitting the area under the curve of the empirical distribution, so that each sub-area is associated with a symbol. In a general way, the memory data follow the normal distribution. we do a statistics on three experimental models and find that abstract information from LSTM layer conform to the hypothesis. Figure 7 plot the statistical distribution of the positive information ξ^+ and negative information ξ^- , for Sentiment Analysis networks. We can see that, all the empirical distributions largely follow the Gaussian distribution (red curves), as shown in Figure 4.

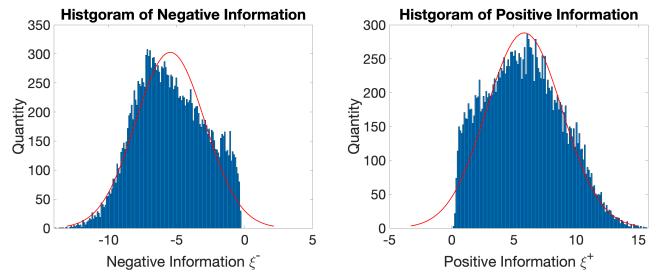


Fig. 7. Statistical distribution of positive and negative information in LSTM sequence [480-484], 2,000 test cases for Sentiment Analysis model are considered.

b) *Selection of Mutation Operations*: We use Algorithm 1 to generate test cases. We detail the mutation operations used for different models. For MNIST model, because the input vector space is continuous, we are able to apply gradient-based mutation, in particular, by configuring $\epsilon \in (0.05, 0.1]$ and $t \in \{1..5\}$.

Different from the MNIST model where a small change to its input image is still a valid image, the input to IMDB model is a sequence of words, on which a small change may lead to an unrecognisable (and invalid) text paragraph. Based on this observation, a gradient-based method is not useful and we take a random-based mutation approach. This is implemented using the EDA toolkit [43], which was originally designed to augment the training data for improvement on text classification tasks. In our experiments, we consider four mutation operations, i.e., \mathcal{M} includes (1) Synonym Replacement, (2) Random Insertion, (3) Random Swap, (4) Random Deletion. The text meanings are reserved in all mutations.

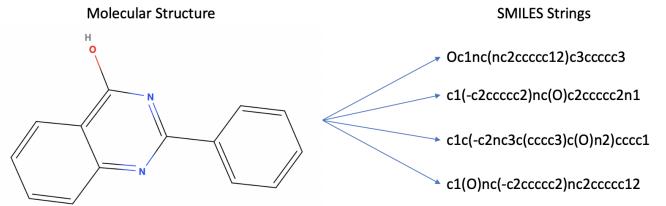


Fig. 8. The same molecular structure represented by different SMILES strings

For Lipophilicity model, we consider a set \mathcal{M} of mutations which change the SMILES string without affecting the molecular structure it represents. The enumeration of possible SMILES for a molecule relies on python cheminformatics package RDkit [35], with the utilization of RDkit, each input SMILES string is converted into molfile format, and then atom order will be changed randomly before converting back. As shown in Figure 8, several SMILES strings can represent the same molecular structure. The enumerated SMILES will be test cases to check if the model really learns the prediction from molecular structure to the lipophilicity logD value, but not just the syntax like SMILES representation.

c) *Oracle Setting*: In our experiments, we set different radius of the test oracle depending on the models. For continuous input problems like MNIST, it's easy to calculate the distance between input images, and we do the experiments (except for those in Section VIII-C5) with $\alpha_{oracle} = 0.005$. For sentiment analysis, it's ambiguous to quantify the text meanings. However, our mutations only apply small perturbations to the original text, with the constraint that the sentiment does not change. The case for Lipophilicity model is similar, we use enumeration approach to generate variational SMILES string of the same molecule, such that obtained SMILES strings have the same molecular structure as the original string.

C. Effectiveness in Testing

In this part, we show the effectiveness of our coverage-guided LSTM testing from the following aspects: (1) the test case generation is efficient, (2) the test suite generated achieved high coverage that is non-trivial, and (3) there is a significant percentage of adversarial examples in the test suite. All experimental results are averaged over 5 runs with different random seeds.

1) *Coverage Improvement by Generated Test Cases*: At first, we compare the coverage results by sampling the 500 inputs from training dataset and by generating 2,000 test cases generated from these samples using Algorithm 1. The Table I, shows the results. We can see that, the coverage level is significantly improved with the test suite generated, and this comes with a non-trivial percentage of adversarial examples.

2) *Neuron Coverage is Easy to Cover*: In our experiments we found that neuron coverage can be trivially achieve. For the MNIST model, on average 35 generated test cases are sufficient to cover all the test conditions. For the IMDB model, it takes 7 test cases on average to reach 100% neuron coverage, and only 20 test cases are needed for the Lipophilicity prediction model. The detailed running results are shown in Figure 9. This shows that neuron coverage is not a suitable metric for RNNs, which also justifies the necessity of our new metrics for LSTM networks.

3) *Gate Coverage and Cell Coverage*: Coverage threshold is a hyper-parameter which needs to be set before the start of testing process. It is interesting to understand how the setting of threshold affects the testing result. In our experiments, for both cell and gate coverage, we sample 1000 test cases and use averaged gate value f_t and averaged aggregate information

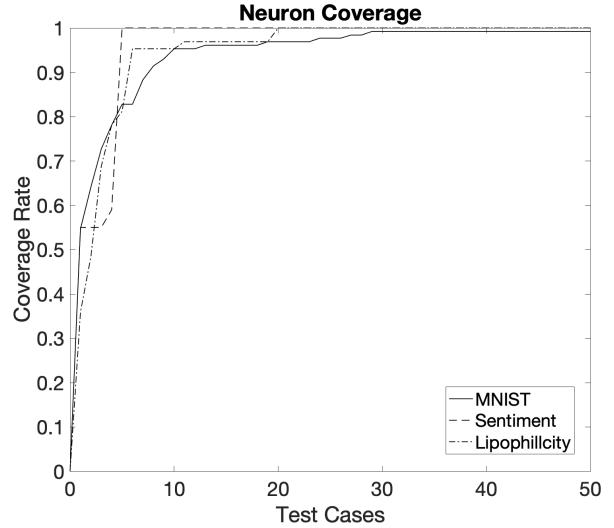


Fig. 9. The updating process of Neuron Coverage in 50 test cases.

change value $\Delta\xi_t$ as the median threshold values, and then gradually decrease and increase them to get a set of experimental settings.

a) *MNIST Model*: we set thresholds $\alpha_\xi \in \{3, 6, 9\}$, $\alpha_f \in \{0.71, 0.78, 0.85\}$ and test the LSTM-2 layer. Since MNIST model has 28 cells in LSTM layer, the total amount of test conditions to cover is 28 for both cell and gate coverage.

b) *Sentiment Analysis Model*: The number of LSTM cells is 500. Thus, there are 500 test conditions for both cell and gate metrics. The threshold values are set as $\alpha_\xi \in \{5, 7, 9\}$ and $\alpha_f \in \{0.73, 0.75, 0.77\}$.

c) *Lipophilicity Prediction Model*: We have $\alpha_\xi \in \{5, 6, 7\}$ and $\alpha_f \in \{0.75, 0.77, 0.79\}$. There are 80 cells in LSTM layer, which means 80 test conditions to be asserted for each metric.

With the addition of mutated samples into our test suite, coverage rate will gradually increase. However, it is common to see that coverage rate may be difficult to improve after reaching a certain level. That is because the remaining test conditions in our coverage metrics can only be asserted by corner cases which are hard to be tested. In our experiments, we record the updating process in 2000 mutants. The plots are shown in Figure 10.

The first impression of coverage updating results is that small value of threshold is less effective in finding adversarial examples. E.g. when we set $\alpha_\xi = 3$ for MNIST, 40 test cases can meet all test conditions. Since test metrics guide the algorithm to terminate, and weak test conditions are easier to be satisfied, less number of test cases are explored. In contrast, if the threshold is assigned with higher value, coverage rate may stay in a relatively low level and be hard to be improved. More test cases are considered, including the corner cases we want to test. Nevertheless, the time consumption will raise for the same goal of coverage rate. And it's very likely that some test conditions may be too strict to meet.

TABLE I
CELL, GATE & SEQUENCE COVERAGE FOR LSTM MODEL TRAINED ON THREE DIFFERENT DATASET

Test Model	Test Cases	# of Adv. Examples	Average Perturbations (L2 norm)	Running Time (in seconds)	Coverage Threshold	Cell	Forget Gate	Coverage Rate Positive Sequence	Coverage Rate Negative Sequence
MNIST	500	0	0	555	$\alpha_h = 6, \alpha_f = 0.78$	0.75	0.77	0.72	0.66
	2,000	592	1.99	4,168	symbols{a, b}	0.89	0.89	0.86	0.84
Sentiment Analysis	500	0	0	726	$\alpha_h = 7, \alpha_f = 0.73$	0.68	0.65	0.91	0.94
	2,000	82	0.11	4,124	symbols{a, b}	0.90	0.91	0.94	0.94
Lipophilicity prediction	500	0	0	236	$\alpha_h = 5, \alpha_f = 0.75$	0.80	0.88	0.94	0.91
	2,000	779	0	949	symbols{a, b}	1.0	0.99	0.94	0.94

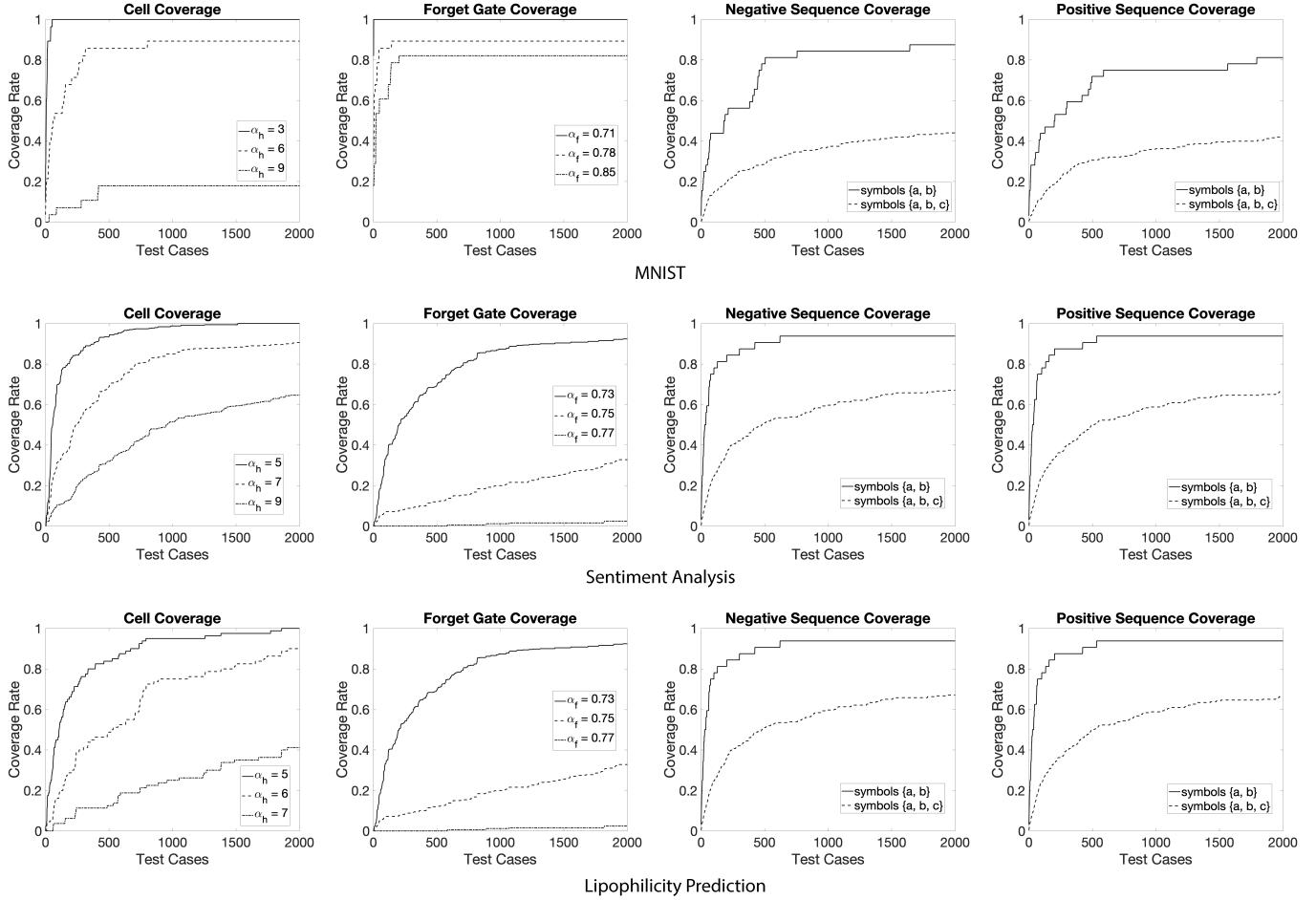


Fig. 10. Illustration of Cell, Gate & Sequence Coverage updating in 2000 test cases For MNIST (28 Cells, Seq. in [19-23], Sentiment Analysis (500 Cells, Seq. in [480-484]) and Lipophilicity Prediction (80 Cells, Seq. in [70,74])

We list experiment results for certain thresholds and compare the results between original seeds input and mutants in Table I. It shows that our mutation-based test generation has the following advantages. First, a significant amount of adversarial examples are found. As stated in Section VIII-A, the three LSTM models have high levels of accuracy in training dataset. The mutation method enable us to search for the misclassified points very close to the training data points. These adversarial data points can be utilized to retrain the model for better robustness. Second, the mutants are in favor of raising coverage rate of a test set. Although there isn't a strong relationship between coverage rate and adversarial examples, the higher coverage rate in a test set with certain amount of

adversarial examples give us more confidence to find various kinds of adversarial examples induced by different root causes.

4) *Sequence Coverage:* Sequence coverage is more complicated than cell and gate coverages, because the number of sequence patterns is exponential with respect to the sequence length. For example, if we test the whole hidden information sequence ξ in MNIST model with 2 symbols {a, b}, There are 2^{28} different patterns to be added as test conditions. Moreover, as discussed earlier, it is likely that a certain portion of patterns are unable to be asserted. The information update inside LSTM is a progressive process which means some hugely-fluctuated patterns are in practice hard to occur in our experiment. In order to simplify the problem and provide a more customized

options, the testing range of LSTM sequence is decided by the users, as the test conditions $\mathcal{R}_{SC}^{[a,b]}$ as given in Equation (6). In our experiments, we focus on the coverage of sequence patterns at intermediate cells, i.e., $[a, b] = [19, 23]$ for MNSIT, $[a, b] = [480, 484]$ for Sentiment Analysis, and $[a, b] = [70, 74]$ for Lipophilicity Prediction.

Our experiments exercise the sequence coverage using 2 and 3 symbols for each model, i.e., $\Gamma = \{a, b\}$ and $\Gamma = \{a, b, c\}$, to represent the sequence data and test the sequential pattern respectively. So the total test conditions to be asserted for 2 symbols are $2^5 = 32$ and for 3 symbols $3^5 = 243$.

The experiment results in Table I and Figure 10 indicates that sequence coverage is a rigorous test requirement. Although we try to cover sequence information inside 5 cells, thousands of test cases are generated. 2 symbols representation shows a good coverage result within 2000 test cases. However, for 3 alphabetic symbols representation, very few sequence patterns are found in consideration of the same test cases. It is expected that if we use more symbols to get more precise representation, coverage rate will be further reduced even with more test cases. On the other hand, this reveals the complexity of LSTM memory. If the sequence coverage is to test the memory of RNNs, sophisticated memory storing mechanism makes the network flexible to make predictions for a large variety of sequential tasks.

5) *The Impact of Oracle on Adversarial examples:* In the above experiments, we fix the oracle radius. However, it is interesting to understand how the setting of oracle, for example the radius, may affect the quantity and quality of the test cases and adversarial examples. Here, we focus on MNSIT model to show how adversarial rate and perturbations on adversarial examples vary with different values of oracle radius. To give more clear comparison, we apply large fluctuations on input images. The hyper-parameter is set with $t \in \{1.5\}$, $\epsilon \in (0.05, 0.3]$, and the range of oracle value is $\alpha_{oracle} = [0, 0.01]$. The experiment results are averaged over 5 different sets of seed inputs. For each set of seeds, 2000 test cases are generated.

As can be seen in Figure 11, the quantity and the perturbations of adversarial examples both increase with the growth of oracle radius. The large oracle radius reveals the weak restrictions imposed on filtering adversarial examples. What's more, it is significant to see that adversarial rate in

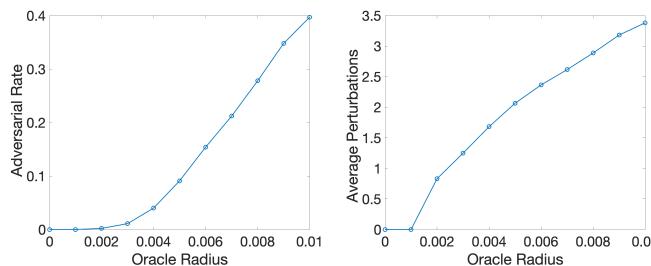


Fig. 11. The number and the average perturbations of adversarial examples against the value of oracle radius

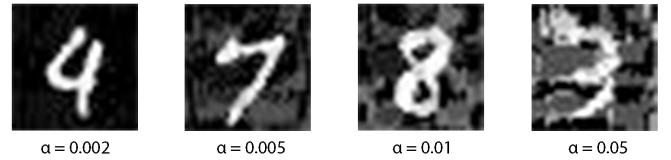


Fig. 12. An illustration of adversarial examples' quality at different oracle radius for MNIST model

the test set goes linearly with oracle radius in an intermediate interval. The average perturbations of adversarial examples are different cases, which show the polynomial relationship with oracle radius. Both figures give the intuitive explanation on how robust the model is. The steeper curve, to some extent, indicates the worse robustness of LSTM to the adversarial examples.

We also plot some adversarial examples for MNSIT in Figure 12 when experiments are set up with different oracle radius. It is expected that large oracle radius can tolerate adversarial examples added with more image noise. The handwritten digit 3 is largely perturbed and even a little indistinguishable by humans. In contrast, digit 4 is under imperceptible noise and much more comply with the definition of adversarial examples.

D. Comparison between Test Metrics

This section explains how test metrics are complementary in covering different functionality of the LSTM layer. We consider minimal test suite, in which the removal of any test case may lead to the reduction of coverage rate. The rationale of taking minimal test suite is to reduce the overlaps, so that it is fairer to compare test metrics. Experimental results for the three models are presented in Table II.

By considering coverage rates in different minimal test sets, it's not difficult to find that the diagonal data is always the largest value in that row and column. E.g., if we derive a minimal test set, which has a very high coverage rate of cell values, it's not necessary to see that other test metrics can achieve a good or even better coverage results. This indicates that our designed test metrics have few interactions in terms of test requirements. Actually, our purpose of developing these test metrics are totally different. As discussed in definitions, cell and gate coverage are one-step test condition, which are developed to test the basic unit of LSTM layer. Nevertheless, sequence coverage brings us back to the multi-step test conditions. When using sequence coverage as stopping conditions for testing, we care more about the information processing progress of the whole LSTM layer.

Another thing should be noticed is that user-specified threshold values for cell and gate coverage or symbols for sequence coverage may affect the comparison results. Although our test metrics are not tightly correlated with each other, weak test requirements are too easy to achieve. That means the minimal test set targeting a test metrics may have relatively good coverage result of another test metric, if the latter ones

TABLE II
COMPARISON BETWEEN TEST METRICS IN MINIMAL TEST SET

Model	Targeted Test Metrics	Threshold / Symbols	Minimal Test Set	Coverage Rate		
				Cell	Forget Gate	Positive Sequence Negative Sequence
MNIST	Cell Coverage	6	21	0.821	0.464	0.219 0.312
	Forget Gate Coverage	0.85	12	0.214	0.821	0.125 0.250
	Positive Sequence Coverage	{a, b}	27	0.357	0.357	0.844 0.344
	Negative Sequence Coverage	{a, b}	27	0.250	0.429	0.219 0.844
Sentiment Analysis	Cell Coverage	7	263	0.908	0.768	0.875 0.812
	Forget Gate Coverage	0.73	262	0.742	0.926	0.875 0.844
	Positive Sequence Coverage	{a, b}	30	0.108	0.078	0.938 0.844
	Negative Sequence Coverage	{a, b}	30	0.108	0.078	0.844 0.938
Lipophilicity prediction	Cell Coverage	6	67	0.900	0.812	0.688 0.817
	Forget Gate Coverage	0.77	29	0.075	0.950	0.469 0.500
	Positive Sequence Coverage	{a, b}	30	0.050	0.637	0.938 0.531
	Negative Sequence Coverage	{a, b}	30	0.075	0.650	0.500 0.938

are weak test conditions. That is reason why in Table II, some minimal test sets depict the high coverage rate for all test metrics.

E. Exhibition of Internal Working Mechanism

We show how to use the generated test cases to understand the LSTM networks. The experiments consider the two models for MNIST and IMDB datasets and visualise their learning mechanism inside the LSTM layer. Figure 13 and 14 present the coverage times for each feature (or test condition). Coverage time counts the number of times a test condition is satisfied in a test suite. For both models, we take a test suite with 2,000 test cases. Coverage time exhibits the difficulty of asserting a feature.

a) *Less Active Features*: For MNIST model, the test suite is obtained by setting thresholds as $\alpha_c = 6$ and $\alpha_f = 0.85$. Note that, each test condition is defined with respect to a row of pixels on the input image. Figure 13 presents two histogram plots for cell and gate, respectively. From the cell plot (left), we can see that the first and last few test conditions are not covered. This is actually expected since we split an MNIST image into rows such that each row is used at a time step. For MNIST images, their top and bottom rows are blank, and therefore the aggregate information change is not significant enough to be over the threshold. The gate plot (right) use a relatively large threshold 0.85. Statistically, in the first several time steps it is hard for the abstract gate value to meet this strict test condition. A reasonable explanation is that the LSTM cell in MNIST model will throw away comparatively more information at the beginning. These unnecessary information is likely the edges of MNIST images, as mentioned above.

For IMDB model, in Figure 14, the plots on the bottom line show the coverage times of all 500 test features for cell and gate coverage, respectively. From the graph, we can see that no matter which coverage metric is considered, in contrast to the last 200 cells, the first 300 cells are less active and obviously “lazy” in trying to be asserted. The reason behind this is that the input samples are padded or truncated to the same length. Since we set the review length to 500 words, the text we used for training and testing maybe too long or too short. The short reviews are padded in the left side with 0s.

Therefore, it's reasonable and likely to see in many test cases, the first few cells are the “dead cells”.

b) *Active Features*: In the cell plot of Figure 13, we can see that cells around 7 and 8 are easy to be asserted, which means that, for many inputs, within this range there are great changes in hidden outputs. In other words, those cells' inputs provide significant information for the classification. In gate plot, the long term memory c_t tends to update lazily at the end of the time series. This can be seen from the high coverage times of the memory values over 0.85 at the last few cells.

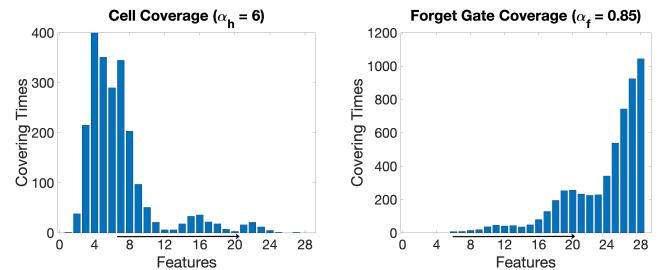


Fig. 13. 2000 test cases are used to demonstrate the coverage times of 28 features in LSTM 2 layer of MNIST model.

c) *Overall Analysis*: If we combine cell and gate plots, the whole working process of LSTM inside the MNIST model becomes transparent. The sequence input of an image starts from the top rows to the bottom rows. At the beginning, the top rows of MNIST images do not contribute to the model prediction, which can be seen from the fact that they do not cause significant hidden state changes. These unimportant information will be gradually thrown away from the long-term memory. When input rows containing digits are fed to the LSTM cells, the model will start learning and the short term memory, represented by the hidden outputs h_t , starts to have strong reactions. The following process will be random due to the diversity of digits. When approaching the end of the sequence, which corresponds to the bottom of digit images, LSTM has already been confident about the final classification and becomes lazy to update the memory. Overall, MNIST digits recognition is not a complicated task for the LSTM model and the top half part of images seems to be enough for the classification.

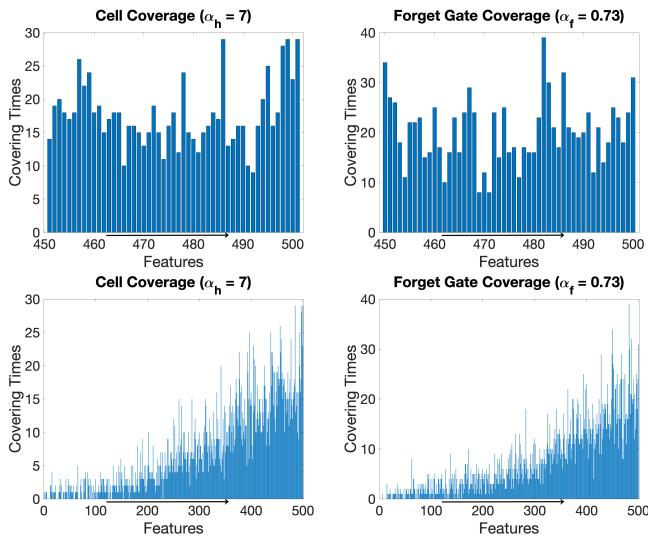


Fig. 14. 2000 test cases are used to demonstrate the coverage times of 500 features in LSTM layer of Sentiment Analysis model.

For IMDB model, each cell plays an important role in dealing with passed information and even affecting the final classification. This is based on the observation of results in Figure 14. We use 2000 reviews, the length of which are all greater than 50. This is to make sure cells between 450-500 have the real input words but not padded 0s. Then the coverage times are counted and plotted which is shown at the top line of Figure 14. Compared with MNIST result in Figure 13, LSTM cells and gates in sentiment analysis model are randomly activated. This can further explain that this network doesn't have a fixed working pattern like MNIST. For a MNIST input, image rows from top to the bottom are gradually changed and combined to get the specific characteristic which can be recognized as the digits we perceive. However, sensitive words in a review text may randomly locate and these words are just model needs to learn for the classification.

IX. RELATED WORK

a) Adversarial Examples for RNN: Adversarial examples represent the major safety/robustness concern for deep neural networks. The adversarial example generation has been an active domain and a large body of research has been conducted for adversarial attacks for recurrent neural networks on tasks such as natural language processing [31], [22], [10], [49], [14], [15], [3] and automated speech recognition [17], [12], [8]. In this paper, adversarial examples are used as a proxy to evaluate the effectiveness of the proposed coverage criteria.

b) Testing feedforward neural networks: Most existing neural network testing methods focus on the feedforward structure. In [32] the neuron coverage is proposed. Though neuron coverage is able to guide the detection of adversarial examples, it is rather easy to construct a trivial test set to achieve very high neuron coverage level. Several refinements and extinctions of neuron coverage are later developed in [26]. Motivated by the usage of MC/DC coverage metrics in high

criticality software, in [37], a family of MC/DC variants are adopted for neural networks, which takes into account the causal relation between features of different layers and closer. It has been formally proved in [37] that the criteria in [32], [26] are specially cases of its MC/DC variant for neural networks.

Besides the structural coverage criteria mentioned above, metrics in [44], [4], [9] define a set of test conditions to partition the input space. Though not being a coverage metric, the method in [25] measures difference measures the difference between training dataset and test dataset based on neural network structural information.

Guided by the coverage metrics, test cases can be generated by neural network testing techniques based on such as heuristic search [48], fuzzing [20], [30], mutation [27], [41] and symbolic encoding [38], [18]. None of the mentioned test generation algorithms has considered RNNs.

c) Testing RNNs: As far as we know, [13] is the only paper so far that provides coverage metrics for RNNs, in which at first an automaton is learned to abstract the RNN under test and then test cases are generated for this automaton. There are major differences between our work and [13]: (1) the approach in [13] treats the RNNs as a black-boxes, whereas our coverage criteria explicitly consider the RNN internal structure; and (2) the metrics in [13] are based on the abstract automaton, instead we define test conditions directly upon the neural network.

d) Visualisation for LSTM: For LSTMs, a sequence of input x_t or hidden state h_t is a sequence of vectors. To get an overview of information inside RNN cells, dimensionality reduction methods (e.g. PCA and t-SNE) have been adopted. For example, [34] employs PCA to extract the most important feature of hidden state at each time step. These methods facilitate the visualisation of RNN hidden behaviors. Our visualisation is completely different, and works by visualising the working principles based on a set of test cases.

X. CONCLUSIONS

In this paper, we develop a test framework for the verification and validation of recurrent neural networks, more specifically networks with LSTM layers. The framework includes a few test metrics to exploit the internal structures of LSTM cells and a test case generation method. Our experimental results show the effectiveness of the test framework in working with a few LSTM networks on different application tasks.

REFERENCES

- [1] Sentiment detection with keras, word embeddings and lstm deep learning networks. <https://www.liip.ch/en/blog/sentiment-detection-with-keras-word-embeddings-and-lstm-deep-learning-networks>, 2018.
- [2] GHTF Study Group 3. Quality management systems - process validation guidance. Technical report, The Global Harmonization Task Force, 2004.
- [3] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani B. Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *CoRR*, abs/1804.07998, 2018.
- [4] Rob Ashmore and Matthew Hill. Boxing clever: Practical techniques for gaining insights into training data and monitoring distribution shift. In *First International Workshop on Artificial Intelligence Safety Engineering*, 2018.
- [5] Alec Banks and Rob Ashmore. Requirements assurance in machine learning. In *AAAI Workshop on Artificial Intelligence Safety (SafeAI2019)*, 2019.

- [6] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, May 2015.
- [7] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), IEEE Symposium on*, pages 39–57, 2017.
- [8] Nicholas Carlini and David A. Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. *CoRR*, abs/1801.01944, 2018.
- [9] Chih-Hong Cheng, Georg Nührenberg, Chung-Hao Huang, and Hiro-toshi Yasuoka. Towards dependability metrics for neural networks. In *Proceedings of the 16th ACM-IEEE International Conference on Formal Methods and Models for System Design*, 2018.
- [10] Minhao Cheng, Jinfeng Yi, Huan Zhang, Pin-Yu Chen, and Cho-Jui Hsieh. Seq2sick: Evaluating the robustness of sequence-to-sequence models with adversarial examples. *CoRR*, abs/1803.01128, 2018.
- [11] J. J. Chilenski and S. P. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193–200, Sep. 1994.
- [12] Moustapha Cisse, Yossi Adi, Natalia Neverova, and Joseph Keshet. Houdini: Fooling Deep Structured Prediction Models. *arXiv e-prints*, page arXiv:1707.05373, Jul 2017.
- [13] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Jianjun Zhao, and Yang Liu. Deepcruiser: Automated guided testing for stateful deep learning systems. *CoRR*, abs/1812.05339, 2018.
- [14] Javid Ebrahimi, Daniel Lowd, and Dejing Dou. On adversarial examples for character-level neural machine translation. *CoRR*, abs/1806.09030, 2018.
- [15] Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-Box Adversarial Examples for Text Classification. *arXiv e-prints*, page arXiv:1712.06751, Dec 2017.
- [16] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. AI2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.
- [17] Yuan Gong and Christian Poellabauer. Crafting adversarial examples for speech paralinguistics applications. *CoRR*, abs/1711.03280, 2017.
- [18] Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S Pasareanu, and Sarfraz Khurshid. Symbolic execution for deep neural networks. *arXiv preprint arXiv:1807.10439*, 2018.
- [19] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning, ICML '06*, pages 369–376, New York, NY, USA, 2006. ACM.
- [20] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. DLFuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 12nd Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2018*, 2018.
- [21] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV2017*, pages 3–29, 2017.
- [22] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *CoRR*, abs/1707.07328, 2017.
- [23] Andrej Karpathy, Justin Johnson, and Fei-Fei Li. Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078, 2015.
- [24] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV2017*, pages 97–117, 2017.
- [25] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. *arXiv preprint arXiv:1808.08444*, 2018.
- [26] Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui Xue, Bo Li, Li Li, Yang Liu, Jianjun Zhao, and Yadong Wang. DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems. In *ASE2018*, 2018.
- [27] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. DeepMutation: Mutation testing of deep learning systems. In *Software Reliability Engineering, IEEE 29th International Symposium on*, 2018.
- [28] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *ACM Conference on Computer and Communications Security*, 2017.
- [29] Yao Ming, Shaozu Cao, Ruixiang Zhang, Zhen Li, Yuanzhe Chen, Yangqiu Song, and Huamin Qu. Understanding hidden memories of recurrent neural networks. In *VAST2017*, pages 13–24, 2017.
- [30] Augustus Odena and Ian Goodfellow. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*, 2018.
- [31] Nicolas Papernot, Patrick D. McDaniel, Ananthram Swami, and Richard E. Harang. Crafting adversarial input sequences for recurrent neural networks. *CoRR*, abs/1604.08275, 2016.
- [32] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. DeepXplore: Automated whitebox testing of deep learning systems. In *SOSP2017*, pages 1–18. ACM, 2017.
- [33] Gábor Petneházi. Recurrent neural networks for time series forecasting. *CoRR*, abs/1901.00069, 2019.
- [34] Paulo E Rauber, Samuel G Fadel, Alexandre X Falcao, and Alexandru C Telea. Visualizing the hidden activity of artificial neural networks. *IEEE transactions on visualization and computer graphics*, 23(1):101–110, 2017.
- [35] RDKit: Open-source cheminformatics. <http://www.rdkit.org>. [Online; accessed 11-April-2013].
- [36] Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131, 2018.
- [37] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Structural coverage metrics for deep neural networks. *EMSOFT2019*, 2019.
- [38] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *ASE*, 2018.
- [39] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR2014*, 2014.
- [40] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019.
- [41] Jingyi Wang, Jun Sun, Peixin Zhang, and Xinyu Wang. Detecting adversarial samples for deep neural networks through mutation testing. *arXiv preprint arXiv:1805.05010*, 2018.
- [42] Yuan Wang, Kirubakaran Velswamy, and Biao Huang. A long-short term memory recurrent neural network based reinforcement learning controller for office heating ventilation and air conditioning systems. *Processes*, 5(3):46, 2017.
- [43] Jason W Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019.
- [44] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *TACAS2018*, pages 408–426. Springer, 2018.
- [45] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideki Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.
- [46] Zhengin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S. Pappu, Karl Leswing, and Vijay S. Pande. Moleculenet: A benchmark for molecular machine learning. *CoRR*, abs/1703.00564, 2017.
- [47] Qian Yang, J. Jenny Li, and David Weiss. A survey of coverage based testing tools. In *Proceedings of the 2006 International Workshop on Automation of Software Test, AST ’06*, pages 99–103, New York, NY, USA, 2006. ACM.
- [48] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. DeepRoad: GAN-based metamorphic autonomous driving system testing. In *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on*, 2018.
- [49] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *CoRR*, abs/1710.11342, 2017.