Automatic RNN Repair via Model-based Analysis

Anonymous Authors[1]

Abstract

Deep neural networks are vulnerable to adversarial attacks. Due to the their black-box nature, it is rather challenging to interpret and properly repair these incorrect behaviors. This paper focuses on interpreting and repairing the incorrect behaviors of Recurrent Neural Networks (RNNs). We propose a lightweight model-based influence analysis, to help understand and repair incorrect behaviors of an RNN. Specifically, we first build an automaton to enable high-quality feature extraction and to characterize the stateful and statistical behaviors of an RNN over all the training data. Compared with the existing techniques on influence function, our method can efficiently estimate the influence of existing or newly added training samples for a given prediction at both sample level and segmentation level. Our empirical evaluation shows that the proposed automaton is able to extract accurate and understandable features. Based on the automaton, our proposed technique could effectively infer the influence instances from not only an entire testing sequence but also a segment within that sequence. Moreover, with the sample-level and segment-level influence relations, our techniques could further remediate the incorrect predictions at the sample level and segment level. To the best of our knowledge, this is the first work that applies influence analysis as a remediation mechanism, offsetting the misclassification made by an RNN.

1. Introduction

In spite of many state-of-the-art applications and high test accuracy, Deep Neural Networks (DNNs) still make mistakes and output wrong predictions. To fix an incorrect prediction, it is important to understand the root cause of the wrong prediction (Koh & Liang, 2017). Once the root cause is identified, users may fix the errors by removing harmful training data or adding specific data to improve model accuracy (Hara et al., 2019). However, due to the black-box nature of the DNN, it is challenging to identify the "most responsible" training samples and explain the wrong predictions. As a result, wrong predictions are difficult to be corrected. Recently, influence functions have been widely studied for interpreting the predictions of DNN by estimating the effect of removing training samples (Koh & Liang, 2017; Khanna et al., 2019; Koh et al., 2019; Hara et al., 2019). However, using it as a method to remediate misclassification or wrong prediction is still challenging.

First, existing influence-function based methods are mostly designed for feed-forward neural networks (FNNs). Given Recurrent Neural Networks (RNNs), they usually suffer from the vanishing gradient and long-distance dependency. As a result, exiting techniques could not be easily applied for RNNs . Second, different from FNNs, RNNs often come with *stateful* structures for processing sequential inputs (*e.g.*, audio, natural language). For a sequential test input, we need to study the effect of its segments more precisely. For example, in automatic speech recognition, we want to identify which training samples are most responsible for the poor recognition of a specific pronunciation (*i.e.*, segment). Existing methods mainly performed influence analysis for the *whole test input* but not at the *segment level*. Third, to use influence analysis based interpretation for repairing a wrong prediction, one needs to select helpful samples from a large number of collected or generated new samples. However, existing influence analysis based methods inevitably introduce intensive computation, making the selection of useful samples very inefficient. As a result, it greatly limits their potentiality to be used as a mechanism to repair the errors of a model.

In this work, we propose a light-weight model-based influence analysis for RNNs. To capture the stateful behaviors of training data, we first construct an automaton from concrete prediction traces of all training data. This automaton extracts accurate features of inputs from RNNs. In our automaton, the state represents the context of input segments and the transition between two states reveals the influential training samples. Given a target test input or a segment of the test input, we could quickly identify the influential

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

training samples from the transitions of the automaton. As part of this work, we also demonstrate the utility of the proposed influence analysis in multiple applications, such as ❶ understanding model behaviors, ❷ fixing influential mislabeled data, ❸ pinpointing Trojan backdoor in an RNN model, and ❹ repairing incorrect predictions.

2. Related Work

Model Extraction. Existing research has developed various approaches to extract DFA (Deterministic finite automaton) from the known RNN architectures. Specifically, early-stage explorations focused on extracting DFA from the second-order RNNs (*e.g.*, (Omlin & Giles, 1996; Giles et al., 1990; 1992)). More recent works extend the preliminary techniques to GRUs and LSTMs, which have higher practicality than the second-order RNNs (*e.g.*, (Weiss et al., 2018; Cho et al., 2014; Chung et al., 2014; Weiss et al., 2019; Okudono et al., 2019; Ayache et al., 2018)). In terms of the state vector partition strategy, existing techniques mainly follow two different methods – (1) equipartition-based approach (Omlin & Giles, 1996; Weiss et al., 2018), which divides each dimension of the latent representations into k equal intervals, and (2) unsupervised learning-based approach (Zeng et al., 1993; Cechin et al., 2003), which applies the existing clustering method (*e.g.*, K-means, GMM) to cluster the state vectors into different groups. As introduced later in Section 3, we follow the second strategy and apply GMM for state partition. Despite using the same partition strategy, our method is fundamentally different from the existing DFA extraction techniques in that none of the existing methods could derive the influence of training samples upon a given testing sample (*i.e.*, influence relations). In addition, to precisely simulate the behaviors of a target RNN, existing methods need to exhaustively search for the state-transitions in the target RNN, which does not scale to RNNs with high-dimensional hidden representations. However, our method only requires a coarse-grained approximation for deriving the influence relationships, which is much lighter-weight than the existing model extraction methods.

Influence Analysis. Koh *et al* (Koh & Liang, 2017) first studied the influence of training samples upon a given testing sample for DNNs. Specifically, they utilized the *influence function* to identify the most representative training samples for a given testing sample. Following (Koh & Liang, 2017), recent efforts have been made to either enable the influence analysis for non-optimal models trained by using non-convex losses (Hara et al., 2019) or analyze the influence of a group of training samples upon a given prediction (Koh et al., 2019). Despite deriving meaningful influence relations for feed-forward networks (*i.e.*, MLP and CNN), the existing methods might not be effective for RNNs due to the infamous gradient vanishing/explosion problem. In this work, our proposed method does not depend on the gradient calculation and could capture the stateful behaviors of the RNN accurately with the automaton. In addition, our method is much lighter-weight as shown in Section 4. Furthermore, our method could provide a finer-grained influence relation than the existing methods or, in other words, we can, at the segment level, pinpoint the most influential training samples to the segments within a testing sample.

Up the the present, little research has been done about locally repairing machine learning model errors. The work that most related to this topic is (Wang et al.), which offsets errors made by logistic regressions by integrating an additional layer into the model to pre-process the error inputs. Different from this work, our method does not modify the model architecture and targets on more complex networks – RNN. It should be noted that our method is different from the techniques about adversarial defenses (Boopathy et al., 2019; Weng et al., 2018; Singh et al., 2018) and noisy learning/data cleaning (Zhang et al., 2018) in that these techniques aim for improving the robustness against adversarial attacks or data poisoning attacks. Whereas, our remediation mechanism locally offsets testing errors of an RNN. In addition to correcting the misclassifications, our technique could also provide explanations.

3. Approach

Overview. At a high level, we first build an automaton to capture the stateful behaviors of all training data. Based on the state transitions of the automaton, we then identify the influential training samples of a given testing input or a segment of the test input. Specifically, to build the automaton, we extract the abstract states by grouping state vectors (*i.e.*, hidden representations of the RNN) of all training data and construct the state transitions based on the transitions of the state vectors (Section 3.1). Then, we construct the transition influence function based on the traces of the automaton and further perform the segment-level and sample-level influence analysis. (Section 3.2). Last but not least, based on the influence analysis, we develop an remediation mechanism to analyze and offset the test errors (Section 3.3).

3.1. Semantic-guided State Abstraction

Definition 1 (RNN) *An RNN is defined as a 5-tuple* $R = (\mathcal{G}_R, d, m, \boldsymbol{h_0}, \mathcal{Y}_R)$*: $\mathcal{G}_R$ is a recursive function* $\boldsymbol{h_t} = \mathcal{G}_R(\boldsymbol{x}_t, \boldsymbol{h_{t-1}})$*, where* $\boldsymbol{h}_t \in \mathbb{R}^d$ *is a d-dimensional state vector,* $\boldsymbol{x}_t \in \mathbb{R}^m$ *is the m-dimensional input vector at time t; d and m are the dimensions of the state vector and the input vector, respectively.* $\boldsymbol{h}_0 \in \mathbb{R}^d$ *is the initial state; The output function* $\mathcal{Y}_R : \mathbb{R}^d \to \mathbb{R}$ *maps an internal state-vector to the output value.*

Given a sequential input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, an RNN generates a sequence of state vectors $(\boldsymbol{h_0}, \boldsymbol{h_1}, \ldots, \boldsymbol{h_n})$ with

the application of $\mathcal{G}_R$. To simplify the notation, we use $\mathcal{G}_R^{\boldsymbol{x}}$ to denote the state vector sequence of the input $\boldsymbol{x}$. $\mathcal{Y}_R$ calculates different types of outputs based on different applications. In this paper, we mainly focus on the classification problem, where the output function maps each state vector to a specific class (*i.e.*, $\mathcal{Y}_R^n : \mathbb{R}^d \rightarrow \{0, \ldots, n-1\}$, n is the total number of classes). Specifically, for the sequence classification problem (*e.g.*, semantic analysis), the output of the last state vector (*i.e.*, $\mathcal{Y}_R^n(\boldsymbol{h_n})$) is the classification result of the whole input sequence. As for the sequence to sequence problem, such as speech recognition, $\mathcal{Y}_R$ transforms each state vector $\boldsymbol{h_i}$ into a character/word in the target language, and all the output characters/words form the translated sentences. It should be noted that different from the feed-forward nets, RNN takes an input sequentially. That is, at each time i, the RNN only processes the current segment $\boldsymbol{x_i}$ of the input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. As such, the influence analysis of RNN is not only to identify the training samples that are most responsible for the whole sample $\boldsymbol{x}$ (*i.e.*, sample-level influence analysis), but also the most influential training samples to a segment of the input $\boldsymbol{x}_i$ (*i.e.*, segment-level influence analysis).

Definition 2 (Influence Automaton) *Given an RNN R and its training data set T, Influence Automaton is a 6-tuple $A = (Q, \sum, q_0, F, \sigma, \mathcal{I})$, where Q is a finite set of states, $\sum$ is the set of alphabet, q_0 is the initial state, F is a set of accepted states, $\sigma : Q \times \sum \rightarrow Q$ are the transitions and $\mathcal{I} : Q \times \sum \rightarrow \mathcal{P}(T)$ is the transition influence function, where $\mathcal{P}(T)$ is the power set of T.*

If the input of the RNN is discrete data (*e.g.*, the text $\boldsymbol{x}$), the word $\boldsymbol{x_i}$ is the symbol of the alphabet. If the input is continuous data (*e.g.*, a sequence of pixels $\boldsymbol{x_i}$ in an image $\boldsymbol{x}$), we could perform the input abstraction that maps $\boldsymbol{x_i}$ to an abstract input $\hat{\boldsymbol{x_i}}$ and treat $\hat{\boldsymbol{x_i}}$ as the symbols of the alphabet.

To help identify influential training samples, the influence automaton should capture the statistical behaviors of the RNN over all the training data. As such, we firstly feed all the training samples in T into the RNN and collect all the state vectors (denoted as $SV = \{\mathcal{G}_R^{\boldsymbol{x}} | \forall \boldsymbol{x} \in T\} \cup \{\boldsymbol{h_0}\}$). Then, a partitioning function $p : \mathbb{R}^d \rightarrow \mathbb{N}$ is applied to group the similar state vectors into one abstract state, which is used as the states of the automaton (*i.e.*, $Q = \{p(\boldsymbol{h}) | \boldsymbol{h} \in SV\}$). Here, the initial state is denoted as $q_0 = p(\boldsymbol{h_0})$. The accepting states are $F = \{p(\mathcal{G}_R^{\boldsymbol{x}}[-1]) | \forall \boldsymbol{x} \in T\}$, where $\mathcal{G}_R^{\boldsymbol{x}}[-1]$ is the last state vector in the output of $\boldsymbol{x}$.

Different from the existing research that extracts automaton only to mimic the prediction of an RNN (Weiss et al., 2018; 2019), our automaton needs to capture the RNN's internal behaviors for the subsequent influence analysis. To efficiently represent a large number of state vectors, we use Gaussian Mixture Models (GMM), an unsupervised clustering method to group the state vectors. The unsupervised clustering requires a pre-specified cluster number K, which directly decides the number of abstract states and thus affects the accuracy of the extracted automaton. However, since there is no explicit ground truth that can be used to measure the correctness of a partition result for influence analysis, it is challenging to find the correct K through cross-validation. To tackle this challenge, we propose a semantic-guided strategy to select an accurate K. The key insight behind is that the state vectors in one group should have similar semantics or, in other words, the RNN should produce a similar output for the vectors in the same group. Based on this insight, we develop a metric to evaluate the partition result and select the K based on the metric. Here, we first introduce *confidence score*, the metric developed to measure the semantics of the abstract state, followed by the selection strategy.

Definition 3 (Confidence Scores) *Given an RNN classifier $R = (\mathcal{G}_R, d, m, \boldsymbol{h_0}, \mathcal{Y}_R)$ and a partition result Q, the confidence score of each state $q \in Q$ is defined as $\boldsymbol{C}_q = [c_0, \ldots, c_{n-1}]$, where*

$$c_i = \frac{|\{\boldsymbol{h} | \boldsymbol{h} \in SV_q \wedge \mathcal{Y}_R^n(\boldsymbol{h}) = i\}|}{|SV_q|}.$$

where SV_q is a set of state vectors of training samples in T that are clustered into the state q, n is the total number of classes for the classifier.

Intuitively, c_i is defined as the ratio of the state vectors in the abstract state q that are predicted as i by the RNN. A high c_i indicates that most of the state vectors, clustered in one abstract state, share similar semantics. Given the confidence score, we further define the state stability as follows:

Definition 4 (State Stability) *For a state q as well as its confidence score $\boldsymbol{C}_q = [c_0, \ldots, c_{n-1}]$, the state is defined as δ-stable, where $\delta = max(\boldsymbol{C}_q)$.*

δ is used to measure the quality of the corresponding group (the abstract state). That is, a high value of δ indicates a well-clustered state. This is because most state vectors in the corresponding abstract state are predicted as the same label $i = \arg\max_{0 \leq i < n} c_i$, indicating this state is influenced by the training samples from the same class. A new input falling into this state is more likely to be classified into i than the other classes. Whereas a low δ indicates that the abstract state is influenced by training samples from different classes. As a result, the RNN will have low prediction confidences for the testing samples belong to this state.

State Abstraction. In addition to stability, we also desire a small number of total abstract states for a better generalizability (Weiss et al., 2018). To achieve these two goals at

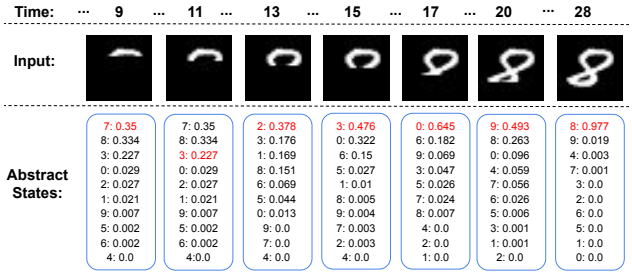


Figure 1: The prediction process of an image and the corresponding abstract states. The highlighted red value shows the model prediction as well as its confidence score of the current input.

the same time, we define the semantics-guided abstraction to decide the K

$$\text{minimize } K, \text{such that } \delta > \theta, \text{where } \delta = avg(\{\delta_{q_1}, \ldots, \delta_{q_n}\}). \tag{1}$$

δ is the average stability of all states (*i.e.* δ), which represents the stability of a partition result. θ is the target threshold of the clustering refinement. A higher θ gives a more stable partition result. Given a pre-specified θ, we increase the cluster size K, starting from 1, and terminate the increment once δ reaches the threshold θ. As is shown later in Section 4, this selection strategy helps to build an automaton that extracts more accurate features.

Figure 1 shows the prediction of an image. At each time, the RNN reads one row from the image and outputs the hidden state. The sequential abstract states as well as the confidence scores are also shown in the third row. In each abstract state, the first column shows the labels and the second column shows the confidence scores. For convenience, the confidence scores are sorted in descending order. We can observe that: 1) except at time 11, all prediction outputs correspond to the largest confidence score in the abstract states; 2) As the prediction confidence of RNN is low when only seeing only parts of the input, it enters into the *non-stable* states in the front. For example, from the human perspective, the input may be 3 at time 13 and 15.

3.2. Light-Weight Influence Analysis

Definition 5 (Trace) *Given an input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, the trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$ is obtained from the state vector $\mathcal{G}_R^{\boldsymbol{x}} = \{\boldsymbol{h}_0, \ldots, \boldsymbol{h}_n\}$, where $q_i = p(\boldsymbol{h_i})$, p is the partitioning function.*

For an input $\boldsymbol{x}$, we first extract a trace that represents its state vector sequence. Based on the abstract states constructed above, we build the transitions and the transition influence function for the influence analysis. Specifically, given the trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$ of each training sample $\boldsymbol{x} \in T$, the transition σ and the transition influence function $\mathcal{I}$ are updated as follows:

$$\begin{aligned} \forall 0 < i \leq n, \sigma = \sigma \cup \{(q_{i-1}, \boldsymbol{x}_i, q_i)\}; \\ \mathcal{I}(q_{i-1}, \boldsymbol{x}_i) = \mathcal{I}(q_{i-1}, \boldsymbol{x}_i) \cup \{\boldsymbol{x}\} \end{aligned} \tag{2}$$

where the influence function $\mathcal{I}$ could capture the effect of training samples at each abstract state.

After updating the influence function based on the state vectors of all the training samples, we perform the influence analysis for a segment of a given test input $\boldsymbol{x}_i$ (*i.e.*, segment-level influence analysis) or the entire testing sequence $\boldsymbol{x}$ (*i.e.*, sample-level influence analysis) using the following methods.

Segment-level Influence Analysis. Given a segment $\boldsymbol{x}_i$ in $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$, we identify the influential training samples of $\boldsymbol{x}_i$ as:

$$infl(\boldsymbol{x}_i) = \mathcal{I}(q_{i-1}, \boldsymbol{x}_i) \tag{3}$$

$infl(\boldsymbol{x}_i)$ represents the set of training samples that have the same segment $\boldsymbol{x}_i$ at the state q_{i-1} and thus are accountable for the prediction of $\boldsymbol{x}_i$. Note that other training samples, which could also include $\boldsymbol{x}_i$ at states other than q_{i-1}, may have low influence or no influence upon the prediction of $\boldsymbol{x}_i$ and thus are not taken as the influential samples of $\boldsymbol{x}_{i-1}$. Taking text as an example, there could be many sentences containing the same word *point*, but with totally different semantics, *e.g.*, '*The pencil has a sharp point*' and '*It is not polite to point at people*'. These training sentences may have very different influences dependent on the test inputs. Our segment-level influence analysis is designed to distinguish such differences and only identify the training samples that are truly influential to a testing segment.

Sample-level Influence Analysis. To quantify the influence of training samples upon an entire testing sequence $\boldsymbol{x}$, we define the temporal feature as follows:

Definition 6 (Temporal Feature) *Given an RNN R, an input $\boldsymbol{x} = (\boldsymbol{x}_0, \ldots, \boldsymbol{x}_n)$, and its trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$, the temporal feature is defined as $\mathcal{F}_{\boldsymbol{x}} = (f_0, \ldots, f_n)$, where $f_i = (ID(q_i), C_{q_i}, \mathcal{Y}_R^n(\boldsymbol{h}_i))$. $q_i = p(\boldsymbol{h}_i)$ is the abstract state to which $\boldsymbol{x}_i$ belongs and $ID(q_i)$ is the unique identifier of the state q_i. C_{q_i} represents the confidence scores (see Definition 3) and $\mathcal{Y}_R^n(\boldsymbol{h}_i))$ is the prediction label at the time i.*

With the definition of temporal feature, we quantify the influence of a training sample on a test input. Specifically, given a training sample $\boldsymbol{x}_{\text{train}}$ and a test sample $\boldsymbol{x}_{\text{test}}$, the influence is quantified as the similarity between the temporal features of the training sample and the testing sample:

$$infl_{score}(\boldsymbol{x}_{\text{train}}, \boldsymbol{x}_{\text{test}}) = \text{similarity}(\mathcal{F}_{\boldsymbol{x}_{\text{train}}}, \mathcal{F}_{\boldsymbol{x}_{\text{test}}}) \tag{4}$$

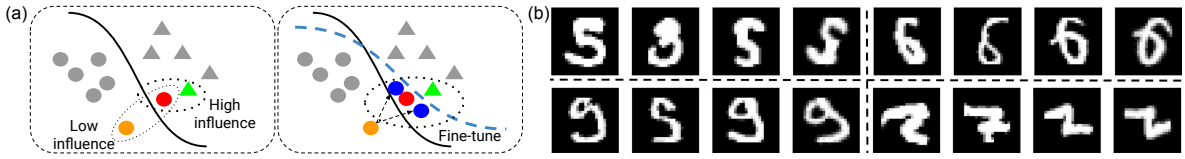


Figure 2: (a) The overview of the fault localization and repair. Red circle represents the failed input. (b) Four examples of data generation for repairing, where each group contains four images (*i.e.*, $\boldsymbol{x}$, $T_{m_{\boldsymbol{x}}}$, $T_{t_{\boldsymbol{x}}}$, $r_{\boldsymbol{x}}$).

The higher the similarity, the higher influence of the training sample $\boldsymbol{x}_{\text{train}}$ upon $\boldsymbol{x}_{\text{test}}$. In other words, due to the high influenced by the training sample $\boldsymbol{x}_{\text{train}}$, the prediction of $\boldsymbol{x}_{\text{test}}$ is very similar to that of $\boldsymbol{x}_{\text{train}}$. Note that different similarity metrics can be selected for different applications. For example, l_p norm distance can be used for the fixed-length input sequences (*e.g.*, image). For the inputs with varying lengths (*e.g.*, natural language texts), one could select the Jaccard distance.

We say that the extracted feature is human understandable because of the extracted confidence (*i.e.*, C_{q_i}) can represent the prediction of the DNN, which is consistent with the human recognition. Considering Figure 1 again, we extract the temporal feature of the input explored by RNN and show it in the third row. Intuitively, the feature is aligned with the human perception. For example, at time 9, the predicted label is 7 and the current input looks like the start of a 7. The confidence score of 7 in the abstract state is not high (0.35). As the input increases, it looks like 3, 2, 3, 0, 9 and 8. At time 17, we can see that it really looks like 0 and the confidence is higher (0.645). Actually, it is still not very high due to that this 0 is not similar to the zeros in training data. At last, it has a very high confidence to predict it as 8. The temporal feature explains the decision process of the RNN on a sequential input.

3.3. Fault Localization and Remediation

With the influence analysis method introduced above, we then develop a remediation mechanism to repair the misclassifications of the target RNN. Specifically, we mainly focus on two kinds of misclassification: 1) misclassification caused by a whole input instance and 2) misclassification caused by an input segment. In the following, we elaborate on our mechanisim of repairing these two different errors.

3.3.1. Remediation with Sample-level Influence Analysis

To repair the first type of errors, we first identify the responsible training samples. Then, we randomly generate new samples by manipulating the identified ones and apply the influence analysis to filter out the error-triggered training samples. Finally, we retrain the target RNN with the newly generated samples.

Fault Localization. Let $\boldsymbol{x}$ be an input misclassified as $m_{\boldsymbol{x}}$ with the ground truth label $t_{\boldsymbol{x}}$, *i.e.*, $t_{\boldsymbol{x}} \neq m_{\boldsymbol{x}}$. By applying the sample-level influence analysis, we first identify the top-n training samples (denoted as $\phi_n^{\boldsymbol{x}}$) that are most responsible for the misclassification of $\boldsymbol{x}$. We use $T_{t_{\boldsymbol{x}}}$ and $T_{m_{\boldsymbol{x}}}$ to denote the training samples in $\phi_n^{\boldsymbol{x}}$, whose ground truth labels are $t_{\boldsymbol{x}}$ and $m_{\boldsymbol{x}}$, respectively. Our key observation is that, *$T_{m_{\boldsymbol{x}}}$ has much more training samples than $T_{t_{\boldsymbol{x}}}$, i.e., the overall influence of $T_{m_{\boldsymbol{x}}}$ is higher than $T_{t_{\boldsymbol{x}}}$* (more detailed results can be found in the supplementary material). This observation explains the reason why $\boldsymbol{x}$ is classified as $m_{\boldsymbol{x}}$. That is, the training samples in $T_{m_{\boldsymbol{x}}}$ have a higher influence upon $\boldsymbol{x}$ than those in $T_{t_{\boldsymbol{x}}}$. The left sub-figure in Figure 2(a) shows an example of the fault localization. The red circle is a test input, which is mainly influenced by the green triangle (*i.e.*, high similarity) than the other circles. As a result, it is misclassified as a triangle.

Remediation. To repair the misclassification, we synthesize new samples whose truth labels are $t_{\boldsymbol{x}}$ but have higher influence on $\boldsymbol{x}$ than the existing training samples. The right sub-figure in Figure 2(a) shows the basic idea of our remediation method. As shown in the figure, we intend to generate new samples (*e.g.*, blue circles) that are more influential than the green triangle. By retraining the model with these synthesized samples, the decision boundary can be fine-tuned such that the misclassified input can be corrected. For a failed input $\boldsymbol{x}$, the samples used for retraining are represented as: $r_{\boldsymbol{x}} = \{\boldsymbol{x'} | \boldsymbol{x'} \in X' \wedge t_{\boldsymbol{x'}} = t_{\boldsymbol{x}} \wedge infl_{score}(\boldsymbol{x'}, \boldsymbol{x}) > \max(\{infl_{score}(\boldsymbol{x'}, \boldsymbol{x}) | \boldsymbol{x'} \in T_{m_{\boldsymbol{x}}}\})\}$, where X' is a set of generated inputs whose truth labels are the same with $\boldsymbol{x}$. The candidate set X' can be generated by multiple techniques (*e.g.*, random augmentation, generative adversarial network). In this work, we synthesize new samples (*i.e.*, X') through data augmentation:

$$X' = \{\boldsymbol{x'} | \boldsymbol{x'} = \text{aug}(\boldsymbol{x''}) \wedge \boldsymbol{x''} \in T_{t_{\boldsymbol{x}}}\}, \tag{5}$$

where aug is the data augmentation technique (*e.g.*, image rotation and shearing). Note that, during the remediation, we do not perform the augmentation on the failed input $\boldsymbol{x}$. Instead, we apply the random augmentations on the training samples in $T_{t_{\boldsymbol{x}}}$, which already have a strong influence upon $\boldsymbol{x}$. Manipulating these samples will be more likely to generate highly influential samples that are more useful for remediation than perturbing other samples. Figure 2(b)

shows some examples of $\boldsymbol{x}$, $T_{m_{\boldsymbol{x}}}$, $T_{t_{\boldsymbol{x}}}$, and $r_{\boldsymbol{x}}$. From the perspective of human perception, in each of the 4 groups, the second image (*i.e.*, $T_{m_{\boldsymbol{x}}}$) looks very similar with the failed input (*i.e.*, $\boldsymbol{x}$). Moreover, after manipulating the third image (*i.e.*, $T_{t_{\boldsymbol{x}}}$), we could get a more influential sample (*i.e.*, $r_{\boldsymbol{x}}$).

3.3.2. Remediation with segment-level influence analysis

Similar with repairing the sample-level error, we also follow a three step procedure to repair the second type of errors. Differently, we design the following method to identify the root cause input segment rather than identifying whole input samples.

Fault Localization Given an input $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ as well as its trace $\tau_{\boldsymbol{x}} = (q_0, \boldsymbol{x}_1, q_1, \ldots, \boldsymbol{x}_n, q_n)$, we identify segments of the input that are more likely to be the root cause of the misclassification as follows:

$$S = \{\boldsymbol{x}_i | 1 \leq i \leq n \wedge |\mathcal{I}(q_{i-1}, \boldsymbol{x}_i)| < \gamma\}$$

where γ is a pre-defined parameter. Intuitively, if the segment $\boldsymbol{x}_i$ has less influential training samples (*i.e.*, less than γ), indicating that the segment $\boldsymbol{x}_i$ is rarely seen under the state q_{i-1} during training, it is more likely to cause the incorrect prediction.

For example, we show one failed input in the sentiment analysis (which is misclassified as negative):

① $\xrightarrow{Just(1,43)}$ ② $\xrightarrow{noticed(1,11)}$ ③ $\xrightarrow{who(1,19))}$ ④$\xrightarrow{gave(1,5)}$ ⑤$\xrightarrow{that(1,89)}$ ⑥$\xrightarrow{out(1,6)}$ ⑦$\xrightarrow{lulz(0,0)}$ ⑧$\xrightarrow{.(0,807)}$ ⑨

The prediction result and the number of the influential training samples are shown after each word. For example, after reading *Just*, 1 represents that the RNN outputs positive. 43 represents that *Just* appears 43 times after the state ① in the training samples (*i.e.*, $|\mathcal{I}(①, Just)| = 43$). We observe that, after the word *lulz*, the RNN returns negative (*i.e.*, 0) because the word *lulz* never appeared after the state ⑦, which thus causes the incorrect prediction.

Remediation To repair the misclassification, we need to insert such segments into the influential training samples such that the missing knowledge (*i.e.*, the appearance of $\boldsymbol{x}_i$ under the state q_{i-1}) could be learned. Specifically, for a localized segment $\boldsymbol{x}_i \in S$, we conduct the remediation with the following steps:

- We randomly select m training samples X_m from $\mathcal{I}(q_{i-1}, \boldsymbol{x}_i)$, where $\forall \boldsymbol{x'} \in X_m, t_{\boldsymbol{x}} = t_{\boldsymbol{x'}}$.
- For each selected training sample $\boldsymbol{x'} \in X_m$, we insert $\boldsymbol{x}_i$ into the corresponding position (*i.e.*, after the state q_{i-1}). Our assumption is that the insertion of $\boldsymbol{x}_i$ will not change the truth label of $\boldsymbol{x'}$ since the selected training sample $\boldsymbol{x'}$ has the same truth label with the failed input $\boldsymbol{x}$ (*i.e.*, $t_{\boldsymbol{x}} = t_{\boldsymbol{x'}}$).
- Finally, we get a set of augmented training samples and train the model to repair the misclassification on $\boldsymbol{x}$.

Table 1: Results of feature analysis (%)

	R_L	ID	(ID, R_L)	CSs	(ID, R_L, CSs)	Ori
MNIST	85.61	80.01	92.35	97.34	**97.50**	98.45
TOXIC	86.62	63.00	87.81	88.90	**89.04**	92.08

4. Evaluation

In our experiments, we evaluated ❶ the correctness of our influence automaton (Sec 4.1), ❷ the effectiveness of our influence analysis (Sec. 4.2) and ❸ the effectiveness of the repair (Sec. 4.3). More evaluation including the Trojan backdoor can be found in the supplementary material.

Datasets and Models. We selected two widely-used public datasets (*i.e.*, MNIST, and Toxic) to evaluate the influence analysis. MNIST (LeCun & Cortes, 1998) is selected for evaluating the sample-level influence analysis by comparing it with the existing baselines. We train an LSTM network with hidden size 100 for this task. At each time, the RNN reads one row (*i.e.*, 28 pixels) from the image. Toxic Comment Dataset (abbrev. Toxic) [1] is selected for evaluating the segment-level influence analysis. The task is to classify whether the comment is toxic or not. We train a GRU network with hidden size 300.

4.1. The Correctness of Automaton and Temporal Features

Setting. The accuracy of the automaton directly affects the influence analysis. As such, we first evaluate the automaton's accuracy by measuring the fidelity of the temporal feature extracted from the automaton. We trained a simple linear classifier (denoted as *SimNN*) with the different components of the temporal features extracted from the training samples and compare their performance with that of the the original RNN. We repeated the experiment 10 times and report the average results in Table 1(a), where column *Ori* shows the test accuracy of the original RNN while other columns use the corresponding temporal features as input.

We can observe that using only *IDs* or *R_L*, *SimNN* achieves a lower accuracy than combing them together on both datasets. With only the confidence scores, the test accuracy reaches 97.34% and 88.90%, much higher than only

[1] https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge.

using *IDs*. It indicates that our semantic-based abstraction captures more information than clustering *IDs*. Finally, models trained with the full temporal features achieve the most comparable performance with the original RNN, which indicates the fidelity of extracted features. Supplementary also shows the results of recovering the original samples from the temporal features, which further validates the correctness of the temporal features.

4.2. Sample-level Influence Analysis for Identifying Influential Mislabeled Training Data

Setting. Similar to the configuration in (Koh & Liang, 2017; Khanna et al., 2019), we randomly mislabeled some training samples and identified such mislabeled samples with influence analysis. Specifically, we first took a subset of MNIST with all the images of digit 1 and 7. Then, we randomly selected 30% images of 7 in the training set, flipped their labels to 1, and trained a binary classifier. We ranked the training samples based on their influence on the test errors of the classifier. We measured the number of mislabeled samples identified (selected based on the influence order) in a certain number of training samples and the number of errors repaired by fixing the identified mislabeled samples. Two state-of-the-art technique – K&L (Koh & Liang, 2017) and SGD(Hara et al., 2019), and the random strategy are selected as the comparison baselines.

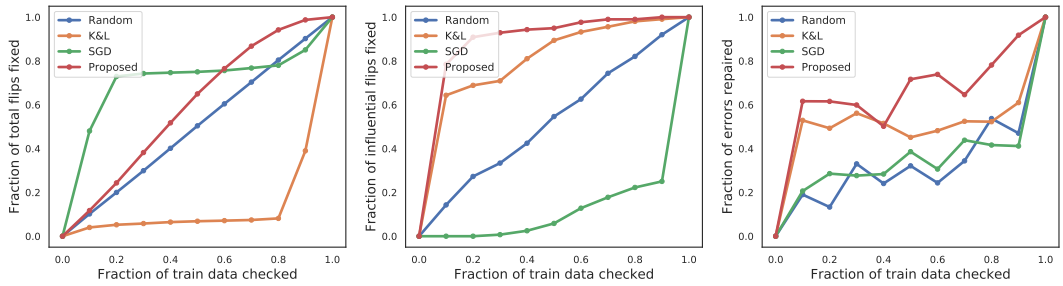
Fig. 3(a) shows the results of identifying flips by checking labels of training samples, following the order of the influence-based prioritization. The horizontal axis represents how many training samples are selected while the vertical axis represents how many flips are identified from the selected samples. Overall, SGD method performs better to quickly identify flips–with the gradient-based estimation, they may identify those training samples that even have only small influence on the loss. However, not all flipped/mislabeled training samples are responsible for the test errors. We found that, although many training samples are mislabeled (*i.e.*, from 7 to 1), most of them are still predicted as 7 after training. Intuitively, such mislabeled samples may have low influence on the errors because they can still be predicted correctly. We consider the mislabeled samples predicted as 1 after training as influential flips. In Fig. 3(b), the vertical axis represents how many *influential* flips are identified in the selected training samples using the influence analysis. The results show that our method and K&L could identify more *influential* flips than the other two approaches. Fig. 3(c) shows the repaired results by fixing all flips in the selected training samples. The results further confirmed that the *influential* flips have more influence on the errors and our method could identify them effectively. However, although SGD identified more flips at an early stage (see Fig. 3(a)), many of them may have lower influence on the errors (Fig. 3(b) and Fig. 3(c)).

Performance. The average running time for automaton extraction is 76.37s, which is a *one-time* cost. Once the automaton is constructed, our influence analysis is very efficient and takes much less time (an average of 1.16s on all errors) than the existing methods (70.13s for K&L and 5690.66s for SGD), indicating that our influence analysis is more scalable than existing techniques.

4.3. RNN Repair via Sample-level Influence Analysis

Setting. We used the MNIST dataset in this experiment. To filter out the errors caused by the randomness, we only select misclassified samples that frequently occur in multiple training runs. Specifically, we trained seven models with different epochs and found 23 commonly failed inputs. Then, we applied random rotation and translation (Engstrom et al., 2019) to these failed samples and obtained a set of candidates. We extracted one automaton from each model and identified the most influential sample from the generated images for each error. As such, we obtained 161 new images and added them to the training set. By using different training epochs, we trained 10 models with the original and augmented training set, respectively. To further knock-off the randomness, we repeated this process 5 times and obtained 50 models from the original and the augmented training set. We compared the accuracy of these models on the 23 failed inputs. We used the random strategy as the baseline, *i.e.*, randomly selecting 161 images from the synthesized images without the influence guidance.

Table 2 summarizes the comparisons among the models trained with the original training data (Row *Ori_Train*), the training set augmented with randomly selected samples (Row *Rand_Train*), and the training data including samples selected by our method (Row *Our_Train*). Column *#Faults* lists the number of errors needed to be repaired. Column *#AvgFixed* shows the average number of errors that are correctly repaired by the 50 models. Column *Distribution of Errors Under the Repair Success Rate* gives the distribution of errors within different repair success rate intervals. Here, the success rate of each error is the percentage of (50) models that could correct it. The results show that our method can effectively repair 50.9% of errors by adding only 161 new training samples. Meanwhile, we can see that these errors are difficult to be correctly predicted using the original training set (only 5.7%) and the training set selected by the random strategy (18.7%). In addition, the repair success rate of the original training set and the randomly selected data are extremely low (*i.e.*, from 0 to 0.2). However, our method performs much better in that it corrected the errors that are consistently misclassified (*e.g.*, 8 and 4 in *Ori_Train* and *Rand_Train*) and overall obtain higher success rates.



(a) Fixing all mislabels (b) Fixing influential mislabels (c) Repairing errors

Figure 3: Comparison on repairing errors by identifying influential mislabeled samples over 10 runs

Table 2: Results of Repairing Erroneous Behavior for RNN on MNIST

	# Faults	#AvgFixed	Distribution of Errors Under the Repair Success Rate						
			0	(0, 0.1]	(0.1, 0.2]	(0.2, 0.5]	(0.5, 0.7]	(0.7, 1)	1
Ori_Train	23	1.3 (**5.7%**)	8	10	5	0	0	0	0
Rand_Train	23	4.3 (**18.7%**)	4	9	3	3	3	1	0
Our_Train	23	11.7 (**50.9%**)	0	3	4	3	7	1	5

4.4. RNN Repair via Segment-level Influence Analysis

Setting. We used the Toxic dataset to evaluate the segment-level repairing. Specifically, we focus on the errors caused by segments, *i.e.*, positive cases predicted as negative rather than negative-to-positive errors that are usually caused by wrong semantics of the whole sentence. Here are some examples:

- **positive-to-negative**: "Who the heck is Ramona anyway ? ? ? ?"
- **negative-to-positive**: " There are rumors that Boss Ross was gay , are there any proof to these claims ? People , wake up ... " I will state here then that she is very pretty "

For the positive-to-negative one, we highlight the word (*i.e.*, heck) that causes the misclassification (after this word, the prediction of the RNN becomes negative). For the negative-to-positive one, it is always predicted as positive during the RNN processing. We observe that even humans are hard to judge it. The key reason is that there is no a clear word that makes it negative. Hence, it is classified as positive.

In addition, some positive-to-negative errors are caused by the un-supported embedding (*i.e.*, the word is embedded as 0) and we ignored such errors. Finally, we selected 23 positive-to-negative test data that are misclassified. For each test case, we set the parameter γ (refer to Section 3.3.2) as 5. To repair such errors, we insert the identified words into some positive sentences in the training data. As a baseline, we use the random strategy to select the same number of sentences for the insertion. Finally, we use the augmented training data for training with 40 epochs (the same with the original model). To mitigate the randomness, we repeat the experiments with 10 seeds.

Table 3: Results of Repairing on Toxic

Num. (m)	5	15	25	35	45
Random	43.63%	63.18%	65.91%	66.36%	61.36%
Our	50%	65.64%	72.73%	81.82%	81.82%

Table 3 shows the results of the segment-based repair. Row *Number* shows the number of training data that are selected for insertion. Specifically, we select 5, 15, 25, 35, 45 training samples (*i.e.*, m in Section 3.3.2) for the augmentation, respectively. Row *Random* and Row *Seg-guided* represent the average success rate of repairing erroneous cases. We can see that, as the number of training samples increases, the success rate also increases. With the random insertion, some errors can be repaired. However, with the segment-influence analysis, we could find the more influential cases that achieve better results.

5. Conclusion

This paper presented a novel model-based technique for influence analysis of RNNs. Different from existing techniques that perform loss change estimation, our method is less computation intensive and more efficient. We could identify the most influential training samples on given test inputs at both segment level and sample level. Based on our RNN influence analysis, we further proposed a method for repairing misclassified samples of RNN. We showed that our techniques are effective in identifying important mislabled training samples, and repairing RNNs. In future work, we plan to improve the GMM-based partitioning with more fine-grained refinement. We also consider introducing more diverse types of data augmentation techniques (*e.g.*, GAN, morphing) to generate candidate data for repairing.

References

Ayache, S., Eyraud, R., and Goudian, N. Explaining black boxes on sequential data using weighted automata. In *ICGI*, 2018.

Boopathy, A., Weng, T.-W., Chen, P.-Y., Liu, S., and Daniel, L. Cnn-cert: An efficient framework for certifying robustness of convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 3240–3247, 2019.

Cechin, A. L., Regina, D., Simon, P., and Stertz, K. State automata extraction from recurrent neural nets using k-means and fuzzy clustering. In *23rd International Conference of the Chilean Computer Science Society, 2003. SCCC 2003. Proceedings.*, pp. 73–78, Nov 2003. doi: 10.1109/SCCC.2003.1245447.

Cho, K., Van Merriënboer, B., Bahdanau, D., and Bengio, Y. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Engstrom, L., Tran, B., Tsipras, D., Schmidt, L., and Madry, A. Exploring the landscape of spatial robustness. In *Proc. of the 36th Intl. Conf. on Machine Learning*, 2019.

Giles, C. L., Sun, G.-Z., Chen, H.-H., Lee, Y.-C., and Chen, D. Higher order recurrent networks and grammatical inference. In *Advances in neural information processing systems*, 1990.

Giles, C. L., Miller, C. B., Chen, D., Chen, H.-H., Sun, G.-Z., and Lee, Y.-C. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 1992.

Hara, S., Nitanda, A., and Maehara, T. Data cleansing for models trained with sgd. In *Advances in Neural Information Processing Systems*, 2019.

Khanna, R., Kim, B., Ghosh, J., and Koyejo, S. Interpreting black box predictions using fisher kernels. In Chaudhuri, K. and Sugiyama, M. (eds.), *The 22nd International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 3382–3390, 2019.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, 2017.

Koh, P. W. W., Ang, K.-S., Teo, H., and Liang, P. S. On the accuracy of influence functions for measuring group effects. In *Advances in Neural Information Processing Systems*, 2019.

LeCun, Y. and Cortes, C. The MNIST database of handwritten digits, 1998.

Okudono, T., Waga, M., Sekiyama, T., and Hasuo, I. Weighted automata extraction from recurrent neural networks via regression on state spaces. *arXiv preprint arXiv:1904.02931*, 2019.

Omlin, C. and Giles, C. Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9 (1):41–52, 1 1996. ISSN 0893-6080. doi: 10.1016/0893-6080(95)00086-0.

Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pp. 10802–10813, 2018.

Wang, H., Ustun, B., and Calmon, F. P. Repairing without retraining: Avoiding disparate impact with counterfactual distributions. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proc. of the 36th International Conference on Machine Learning (ICML), 2019*.

Weiss, G., Goldberg, Y., and Yahav, E. Extracting automata from recurrent neural networks using queries and counterexamples. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 5247–5256, 2018.

Weiss, G., Goldberg, Y., and Yahav, E. Learning deterministic weighted automata with queries and counterexamples. In *Advances in Neural Information Processing Systems*, pp. 8558–8569, 2019.

Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Boning, D., Dhillon, I. S., and Daniel, L. Towards fast computation of certified robustness for relu networks. *arXiv preprint arXiv:1804.09699*, 2018.

Zeng, Z., Goodman, R., and Smyth, P. Learning finite state machines with self-clustering recurrent networks. *Neural Computation*, 5, 11 1993. doi: 10.1162/neco.1993.5.6.976.

Zhang, X., Zhu, X., and Wright, S. Training set debugging using trusted items. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.