Efficient Adversarial Sequence Generation for RNN with Symbolic Weighted Finite Automata

Mingjun Ma, Dehui Du, Yuanhao Liu, Yanyun Wang, Yiyang Li

Software Engineering Institute
East China Normal University, Shanghai, China
dhdu@sei.ecnu.edu.cn

Abstract

Adversarial sequence generation plays an important role in improving the robustness of Recurrent Neural Networks (RNNs). However, there is still a lack of effective methods for RNN adversarial sequence generation. Due to the particular cyclic structure of RNN, the efficiency of adversarial attacks still need to be improved, and their perturbation is uncontrolled. To deal with these problems, we propose an efficient adversarial sequence generation approach for RNN with Symbolic Weighted Finite Automata (SWFA). The novelty is that RNN is extracted to SWFA with the symbolic extracting algorithm based on Fast k-DCP. The symbolic adversarial sequence can be generated in the symbolic space. It reduces the complexity of perturbation to improve the efficiency of adversarial sequence generation. More importantly, our approach keeps perturbation as much as possible within the human-invisible range. The feasibility of the approach is demonstrated with some autonomous driving datasets and several UCR time-series datasets. Experimental results show that our approach outperforms the state-of-art attack methods with almost 112.92% improvement and 1.44 times speedup in a human-invisible perturbation.

Introduction

While Recurrent Neural Networks (RNNs) have made great breakthroughs in safety-critical domains in recent years, such as autonomous driving (Li et al. 2020) and surgical robots (Zhou et al. 2020). Such AI-powered applications are facing the challenge of trustworthiness because of its vulnerability to adversarial attacks (Szegedy et al. 2013a).

However, most of the adversarial generation algorithms (Szegedy et al. 2013a; Goodfellow, Shlens, and Szegedy 2014; Wei, Guo, and Li 2021) designed for Feed-forward Neural Networks (FNNs) are not very good at handling RNNs, which has a particular cyclic structure. When dealing with the input data in the form of a long sequence, traditional adversarial example generation algorithms face two challenges. The first challenge is that the sequence data usually reflects more trend information compared with the image data, thereby the perturbations on sequences are easier to perceive, which increases the difficulty of generating adversarial sequences. The second one is that, as the sequence length increases, the complexity of the perturbation will also increase exponentially. This ultimately makes it more difficult to craft adversarial examples on sequential data.

Another attempt to improve the robustness of RNN is learning a simpler interpretable model such as finite automata from the trained RNN (Omlin, Thornber, and Giles 1996; Zhang et al. 2021). Among those automata models, Symbolic Weighted Finite Automata (SWFA) (Suzuki et al. 2021) is the most promising model because it can mimic RNN to perform real-value operations and accept infinite input. However, the existing work about extracting SWFA from RNN is relatively lacking, and the ability of SWFA to perform real-value operations has not been fully explored. Many valuable properties have yet to be explored, such as model checking (Clarke et al. 2018) and adversarial attacks on that.

In summary, adversarial attacking methods still suffer from their low efficiency on the RNN with complex cyclic structures. In order to deal with this problem, we propose an efficient adversarial sequence generation approach with the SWFA extracting algorithm. The aim is to utilize the symbolic abstraction technique to improve the efficiency of adversarial sequence generation and control the size of perturbation within the human-invisible range.

Our main contributions can be summarized as follows:

- An efficient adversarial sequence generation approach for RNN by SWFA is proposed. The key is to extract SWFA from RNN with the symbolic extraction algorithm.
- The proposed adversarial sequence generation algorithm has been implemented, which outperforms the state-of-art attack methods with almost 112.92% improvement and 1.44 times speedup.
- The experiments on the autonomous driving dataset show that the adversarial sequences generated by our approach are more covert compared with the state-of-the-art methods.

Related Work

There have been many efforts made to improve the robustness of RNN, and adversarial example generation is one of the most popular methods. Depending on how much information an adversary can access, adversarial attacks can be

classified into two categories: white-box attack (Szegedy et al. 2013b; Goodfellow, Shlens, and Szegedy 2015) and black-box attack (Kurakin, Goodfellow, and Bengio 2016; Chen et al. 2017). As for whether the adversarial attack algorithms applied in the FNN are effective on RNN, it has been discussed in (Kurakin, Goodfellow, and Bengio 2016). However, the research on adversarial example generation optimized for RNN is rare, which is still a challenging problem.

Besides, the extracting abstract models from RNN has attracted more attention recently. Most are focused on extracting Label Transaction System (LTS) (Gorrieri 2017), such as DFA (Omlin, Thornber, and Giles 1996), WFA (Ayache, Eyraud, and Goudian 2018; Weiss, Goldberg, and Yahav 2018; Zhang et al. 2021), DTMC (Du et al. 2019), and PFA (Dong et al. 2019). But the existing work aims at outputting boolean values, which is not suitable for the real-world RNN applications. This challenge motivates the extraction of the quantitative finite state machine as an abstraction of RNN. WFA is a powerful abstract model for RNN because it outputs the real value. For WFA extraction work, there are spectral techniques (Ayache, Eyraud, and Goudian 2018), L* query learning algorithm (Weiss, Goldberg, and Yahav 2018), and Decision-guide approach (Du et al. 2019).

This work (Suzuki et al. 2021) proposes the query learning algorithm for Symbolic Weighted Finite Automata (SWFA). As an extension of WFA, SWFA can handle strings over infinite alphabets more efficiently, which generalizes WFAs by allowing transitions to be functions from a possibly infinite alphabet to weights. Inspired by their work, we propose Fast k-DCP to extract SWFA from RNN, which can be utilized to improve the efficiency of adversarial sequence generation.

Preliminaries

This section briefly introduces some necessary preliminaries and notations to understand our approach. Specially, we summarize the essence of the adversarial sequence generation approach and the symbolic abstracting technology.

Recurrent Neural Network

Recurrent Neural Network is denoted as a recursive function $h^{(t)}(\vec{x}) = f(h^{(t-1)}(\vec{x}), \vec{x}^{(t-1)})$, representing the state transitions during the learning process. $h^{(t)}(\vec{x}) \in \mathbb{R}^n$ is the hidden state at time t, which is calculated by the input at time t and the hidden state at time $t-1$ together. RNN can also be formulated as a tuple (Michalenko et al. 2019).

Definition 1 (Recurrent Neural Network) *RNN is denoted as a 6-tuple $R = (H, X, Y, h_0, f, g)$, where*

- *H denotes the set of hidden states.*
- *X is the possible input space.*
- *Y is the final output of RNN.*
- *h_0 is the initial hidden state.*
- *$f : H \times X \to H$ denotes the transition function.*
- *$g : H \to Y$ denotes the output function.*

Note: During the entire RNN operation, f is called repeatedly, and g is called only once for the last output. And for classification tasks, g is specified as a softmax function. f and g are shared between different time steps t in an RNN.

As for the input data, we define the dataset with n samples as $D = \{(x_i, l_i) \,|i = 1, 2, \ldots, n\}$, where $x \in X$ is an input sample and l is the true label of the sample. For classification tasks, l is in a finite set L containing all possible labels. A single sample $x \in X$ with m-sequence is in the form like a vector $x = [x^{(1)}, x^{(2)}, \ldots, x^{(i)}, \ldots, x^{(m)}]$, which includes the input at different time steps. $x^{(i)}$ is the input value at the i-th time step on sample x. Input space matrix $X \in \mathbb{R}^{n \times m}$, where n denotes the number of samples and m indicates the sequence length. The matrix $Y \in \mathbb{R}^{n \times |L|}$ is used to store the final output, which is the possibility of each label for each sample in an RNN classification task.

Symbolic Weighted Finite Automata

A Symbolic Weighted Finite Automata (SWFA) Υ over $\mathcal{G}$ can be formulated as a tuple (Suzuki et al. 2021).

Definition 2 (Symbolic Weighted Finite Automata) *SWFA is denoted as a 5-tuple $\Upsilon = (\mathcal{G}, Q, \alpha, \beta, A)$, the set of all functions from X to Y is denoted by Y^X.where*

- *$\mathcal{G}$ is a class of guard functions from an alphabet Σ to a semiring $\mathbb{S}$.*
- *Q denotes the finite set of states.*
- *α is a vector of the initial weights $\in \mathbb{S}^Q$.*
- *β is a vector of the final weights $\in \mathbb{S}^Q$.*
- *A denotes a transition relation $\in \mathbb{G}^{Q \times Q}$, A_σ represents the transition relation for $\sigma \in \Sigma$, $A_\sigma[q, q'] = A[q, q'](\sigma)$ for all $(q, q') \in Q^2$. A_σ can be extended for strings $w \in \Sigma^*$ as $A_w = A_{\sigma_1} A_{\sigma_2} \ldots A_{\sigma_l}$, where $w = \sigma_1 \sigma_2 \ldots \sigma_l$, with $\sigma_i \in \Sigma$. The formal power series f_Υ represented by Υ is defined by $f_\Upsilon(w) = \alpha^T A_w \beta$ for all $w \in \Sigma^*$.*

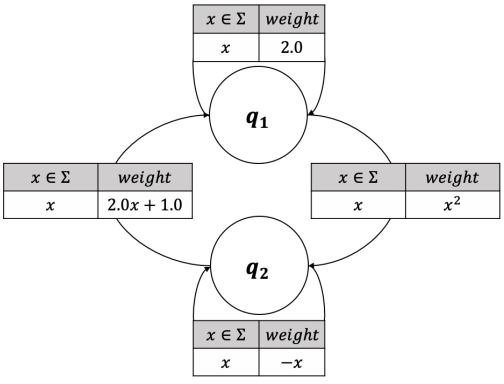

Figure 1: SWFA with $Q = \{q_1, q_2\}$ and $\Sigma = \{-1, 0, 1\}$

SWFA can deal with a possibly infinite alphabet efficiently taking advantage of the structure of the alphabet. The minimization and equivalence checking for SWFAs can be achieved.

Example 1 Let $\Sigma = \{-1, 0, 1\}, Q = \{q_1, q_2\}$, $\alpha^T = [1 \quad 0]$, $A = \begin{bmatrix} 2.0 & x^2 \\ 2.0x + 1.0 & -x \end{bmatrix}$, $\beta = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}$. If the input is $x = [1, -1]$, the result is obtained by multiplying a series of matrices as follows.

$$\alpha^T \cdot A_{-1} \cdot A_1 \cdot \beta = [5 \quad 3].$$

And such an SWFA can also be presented in a graphical view, as shown in Fig.1. Each circle represents a state q, the label on each edge from q to q' is the assigned guard function from Σ to $\mathbb{Q}$, which represents its symbolic nature.

As we can see, the symbolic representation of the weight on transitions enhances the abstraction ability of WFA. It is very suitable to model the infinite alphabet of RNN's input.

Adversarial Example Generation

Adversarial example generation is a technology that uses adversarial attack methods to generate adversarial examples. An adversarial example refers to a sample formed by artificially adding subtle perturbations that are invisible to the naked eyes. Such samples will cause the trained model to misclassify the output (Huang et al. 2020). An adversarial example can be denoted as:

$$\begin{aligned} \arg\min_i \mathbb{H}_i(x') = \arg\min_i \mathbb{H}_i(x) \\ \wedge ||x - x'||_p \leq d \\ \wedge \arg\max_j f_j(x') \neq \arg\max_j f_j(x) \\ s.t. \quad p \geq 1, p \in N, d > 0. \end{aligned}$$

$\mathbb{H}$ stands for the human inception, and f denotes the deep neural network's judgment. Within the subtle perturbation d, the model makes the wrong classification. An adversarial example has two characteristics, one is that the perturbation is so small that humans cannot notice it easily. The other is that the classification result may be wrong.

Researchers (Szegedy et al. 2013a) have formulated the search for adversarial examples as an optimization problem to find the minimum ϵ, and used box-constraint L-BFGS to conduct attacks.

$$\arg\min_\epsilon f(x + \epsilon) = l, \ s.t. \ (x + \epsilon) \in D$$

The input $x + \epsilon$ is in the same domain D with x, but gets a different label, which is an adversarial example. Though the above optimization solution has a good performance, it is computationally cost. Then an efficient adversarial attack algorithm was introduced (Goodfellow, Shlens, and Szegedy 2014). It is named as Fast Gradient Sign Methodology (FSGM) which utilizes the gradient of the cost function to generate the adversarial examples.

Besides L-BFGS and FGSM, most of the adversarial attack methods are extended based on iterative methods to generate adversarial examples efficiently.

Main Approach

In this section, we present the details of our adversarial sequence generation approach. Fig. 2 shows the overall framework. There are two phases in our approach.

Symbolic Weighted Finite Automata Extraction. The extracting algorithm extracts SWFA from the trained RNN. It is the key part of the approach, which constructs the symbolic state space. On the other hand, the symbolic input are abstracted from datasets.

Adversarial Sequences Generation by SWFA. We perturb the symbolic input to get symbolic sequences. To accelerate the generation process, we formally analyze the reachability of SWFA to rapidly check whether the newly generated symbolic sequence is an adversarial one. If the symbolic sequence is judged as an adversarial symbolic sequence, we can collect concrete adversarial sequences from symbolic adversarial sequences by sampling techniques.

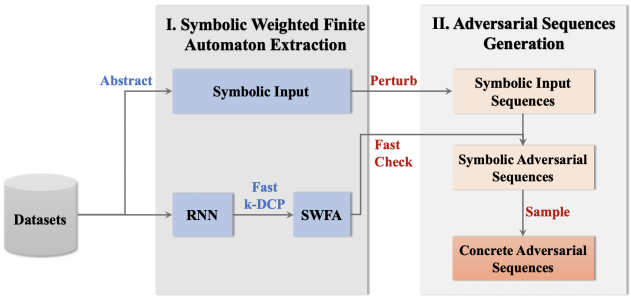

Figure 2: The Overall Framework of Our approach

Symbolic Weighted Finite Automata Extraction

Definition 3 (Fast k-DCP) *Fast k-DCP is adapted from k-DCP (Du et al. 2019), which is used for SWFA extraction from RNNs. k-DCP of a probability vector captures the top k ranked class labels as well as their prediction confidence levels. Compared with k-DCP, Fast k-DCP has the following new features:*

- *High Efficiency: Different from k-DCP, Fast k-DCP discards the time-consuming k-means clustering process and divides hidden states of RNN into different symbolic blocks directly.*
- *Symbolic Abstraction: We extend Fast k-DCP for the infinite alphabet, which deals with input symbolically.*

The cost of symbolic abstraction will increase as the complexity of the learning task increases. It is not cost-effective to generate a small batch of adversarial examples when the task is complex; it has advantages in a scenario where the task complexity is lower and a large number of adversarial examples need to be generated. Assuming that the number of samples is n, the sequence length is m, the feature dimension is s, and the time complexity of symbolic abstraction can be denoted as $O(mns)$, as the complexity of the task increases, which means that n, m, or s becomes larger, the time cost of symbolic abstraction will also increase. The space complexity of symbolic abstraction is $O(T^s)$. As the feature dimension s increases, the space consumption will also grow. But when the complexity of RNN increases, its abstraction cost will not increase accordingly. The abstraction cost is only related to the scale of the task. Therefore,

Algorithm 1: RNN-SWFA by Fast k-DCP

input : RNN $R = (H, X, Y, h_0, f, g)$
Input sequences W
K, T
output: SWFA $\Upsilon = (\mathcal{G}, Q, \alpha, \beta, A)$

1 Initialize $Q' = [s_0], \Sigma' = [\,], Q = [\,], \Sigma = [\,]$;
2 Initialize $A = [\,], \alpha = \pi q_0, \beta = [\,]$;
3 $d = ComputeDistance\,(W)$;
4 $T_{input} = \lceil 10/(d)/(|W| \times |w|) \rceil$;
5 **for** $w \in W$ **do**
6 $\quad s = [h^{(i)}(w)]_{i=0}^{|w|}$;
7 $\quad$ **for** $i = 1$ **to** w **do**
8 $\quad\quad Q'.add(s_i)$;
9 $\quad\quad \Sigma'.add(w_{i-1})$
10 **for** $q' \in Q'$ **do**
11 $\quad Q.add(fkdcp^{K,T}(q'))$;
12 **for** $\sigma' \in \Sigma'$ **do**
13 $\quad \Sigma.add(fkdcp^{|\sigma'|,T_{input}}(\sigma'))$;
14 **for** $\sigma \in \Sigma$ **do**
15 $\quad A_\sigma = BuildTransitionMatrix(\sigma)$;
16 $\quad A.add(A_\sigma)$;
17 **for** $q \in Q$ **do**
18 $\quad \beta_q = 0$ with length $|L|$;
19 $\quad$ **for** $q' \in Q'$ **do**
20 $\quad\quad$ **if** $fkdcp^{K,T}(q') == q$ **then**
21 $\quad\quad\quad \beta_q[argmax(g(q'))] += 1$;
22 $\quad \beta_q = \beta_q / \sum(\beta_q)$;
23 $\quad \beta.add(\beta_q)$;
24 $\mathcal{G} = GuardFuntionLearning(Q, \alpha, \beta, A)$;
25 **return** SWFA $\Upsilon = (\mathcal{G}, Q, \alpha, \beta, A)$

building SWFA has a comparative advantage in such a scenario.

The proposed algorithm for extracting SWFA from RNN using Fast k-DCP is elaborated in Alg 1. It takes the target RNN R, the input sequences W which is a set of word w, the hyper-parameters K and T as inputs, and generates the extracted SWFA.

To get the symbolic input, we calculate T_{input} (Line 3 to Line 4), and then use Fast k-DCP to abstract the input (Line 12 to Line 13) into symbolic blocks. Another purpose of calculating T_{input} is to control the perturbation in a range that is invisible to the human eye. It is a covert adversarial example. And we strongly recommend that $computeDistance$ uses the Euclidean distance L_2 as the distance metric since the ideal task for our approach to show the advantage of efficiency is the one whose dimension of input data is relatively not too high, at which Euclidean distance has been found to capture proximity more accurately (Gopinath et al. 2017).

We execute R on the input sequences W and record the hidden state traces as s (line 5 to Line 9). By this way, we collect the step-wise configurations in a set of hidden states Q', as well as the alphabet Σ'.

By applying Fast k-DCP $fkdcp$ twice (Line 10 to Line 13), we construct the set of abstract states Q and the symbolic alphabet Σ of SWFA. The symbolic alphabet is abstracted from the complete input, that is, the parameter K of $fkdcp$ here is set to the dimension of the input $|\sigma'|$.

For each symbolic input σ in Σ (Line 14 to Line 16), we use $BuildTransitionMatrix$ to count transition frequency among abstract states and then build the transition matrix A_σ for σ. All the matrices are collected into A.

At last (Line 17 to Line 23), the initial distribution over the abstract states would be set to the one-point format, that is, the initial state α of SWFA. And we build the final vector β by calculating β_q for each state. And $GuardFuntionLearning$ learns the $\mathcal{G}$ from Q, α, β, A (Suzuki et al. 2021). In short, extracting SWFA from RNN is the main novelty of our approach. It utilizes the symbolic state models to imitate the learning process of RNN. It helps to generate the adversarial sequence effectively.

Adversarial Sequence Generation

In order to generate adversarial sequences efficiently, we take full advantage of SWFA's symbolic feature to perturb its symbolic input. It reduces the complexity of perturbation. Another novelty of our approach is to abstract the input sequence with the symbolic technique. It is another organizational scheme that abstracts original input into several symbolic blocks.

Definition 4 (Symbolic Block) *Symbolic Block is denoted as a 3-tuple $B_{symbolic} = (X, K, T)$, where*

- *The parameter X means the input sequences.*
- *K denotes the dimension of the symbolic block. (Usually, K is less than 5)*
- *The parameter T denotes the granularity of the symbolic block.*

The input is abstracted into several regions by Fast k-DCP, and the regions can be composed of rectangles, cubes, or hypercubes according to K. Thus these equally divided regions containing multiple real values are called as Symbolic Blocks. It's used to abstract the input space.

Example 2 Assuming the input is normalized into an interval [0,1], we equally divide the input into T parts. As shown in Fig. 3, there are $3 \times 3 \times 3$ symbolic blocks where $K = 3$ and $T = 3$. Each symbolic block is like a small cube, and several symbolic blocks construct a vector space with a total length of 1. Since the length of the vector is fixed, the process of abstracting input into symbolic input is very efficient and not time-consuming. We can quickly find another similar input given a particular input based on the symbolic block.

We take Manhattan distance (L_1) as the abstract distance metric to measure the distance of the symbolic input, which is shown as the following equation. x^* denotes the symbolic input.

$$||x^{*'} - x^*||_1 = \sum_{i=1}^{n} |x_i^{*'} - x_i^*| \tag{1}$$

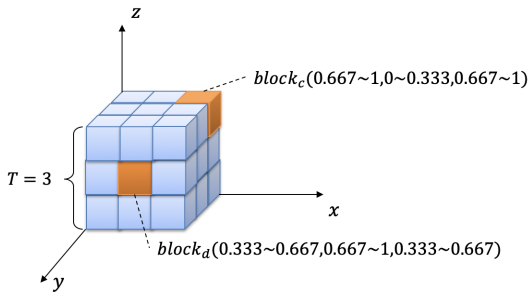

Figure 3: Abstract input space is divided into $3 \times 3 \times 3$ Symbolic Blocks with $T = 3$, $K = 3$

The metric value is assumed as an integer when expressing the distance of the symbolic input for the convenience of the later perturbation and guiding the value of T. Though L_1 and L_∞ meet this requirement, L_∞ is not suitable due to its poor experimental results, so we choose L_1 for the symbolic distance measurement. With the help of the distance metric of the symbolic input, we can reduce the complexity of perturbation.

The difference between the abstract perturbation and the concrete perturbation is shown below.

$$||x' - x||_2 = \epsilon \tag{2}$$

$$||x^{*'} - x^*||_1 = \epsilon^* \tag{3}$$

Equation (2) means that under the L_2 (Manhattan distance) metric, the distance between the disturbed sample and the original sample is denoted as ϵ. Equations (3) describes the distance between the symbolic input x^*. Its perturbed value $x^{*'}$, measured by L_1, and ϵ^* denotes the abstract perturbation. Since we have $\delta = \frac{1}{T} * (Data_{\max} - Data_{\min})$, and the input is normalized in the early stage. We get $D_{\max} = 1$, $D_{\min} = 0$, so the relationship between the abstract perturbation and the concrete perturbation can be denoted as $\delta = 1/T$, which is also the unit of abstract perturbation.

Configuration of T_{input} It is very important to set an appropriate T_{input} to get the symbolic input, which determines the granularity of symbolic blocks and controls the abstract perturbation within the human-invisible range as expected. The value of T_{input} should satisfy the following property.

$$\delta = \frac{1}{T_{input}} \leq \frac{\sum_{i=2}^{n} |x_i - x_{i-1}|}{n-1} \tag{4}$$

$$T_{input} \geq \frac{n-1}{\sum_{i=2}^{n} |x_i - x_{i-1}|} \tag{5}$$

Perturbation and Instantiation In this part, we will iteratively increase ϵ^* to search for symbolic adversarial sequence and instantiate them to concrete adversarial sequences by sampling techniques, such as random sampling. Based on the concept of Symbolic Block and Abstract Distance Metric, we can perturb the sequences (as shown in Fig. 4)in an efficient way. The main idea is shown in Alg.2.

The first step as shown (Alg.2, line 3) is to find the nodes that have the greatest impact on the sequence classification

Algorithm 2:
Adversarial Sequence Generation by SWFA

input : Input x_0;
Continuous $c \in \{true, false\}$;
Number of nodes to be disturbed n;
Perturbation intensity $\epsilon^* \in \{1, 2, 3\}$.
output: Adversarial Sequence x'.

1 Initialize $x_0^{*'} = fkdcp(x_0)$; $x^{*'} = []$; $\delta = 0$;
2 **while** $\delta < \epsilon^*$ **do**
3 $Nodes = NodesSearch(c, n)$;
4 **for** $node \in Nodes$ **do**
5 $dir = FindDirectionbyImportance(x_0^{*'})$;
6 $x_{pert} = Pert(x_0^{*'}, node, dir, \delta)$;
7 **if** $verifyBySWFA(x_{pert})$ **then**
8 $x^{*'}.add(x_{pert})$;
9 $\delta = \delta + 1$;
10 $x' = Sampling(x^{*'})$;
11 **return** x'

results. By exploring several experiments, we find an interesting phenomenon that we call it as SWFA's **000-status**.

Definition 5 (000-status) *000-status is denoted as a vector ζ, where*

- *ζ is a vector of SWFA's intermediate weights $\in \mathbb{S}^Q$.*
- *Each of the components of the vector ζ is 0.*

It represents that the intermediate status of SWFA is like $[0, 0, ..., 0]$ and the input exceeds the generalization limit of our model. A lot of experimental results show that samples with **000-status** are more fragile and more likely to be adversarial sequences. The interesting results inspire us to use SWFA to calculate the intermediate status of the entire sequence at each point. And we record the **000-status** point as the perturbation point.

The second step is to find the appropriate perturbation direction by the least importance denoted by $P^i_{importance} = S^i/(m \times n)$, where superscript i represents the index of the block and S means the number of sequences that fall into the block. We use the insight above, if **000-status** appears in a certain perturbation direction in the SWFA computation result, then these directions should be traversed.

In the third step, the value of ϵ^* is iteratively increased to conduct the abstract perturbation. The size of ϵ^* represents the intensity of the perturbation. We also provide the parameter c to control whether the perturbation is continuous.

Symbolic Adversarial Sequence Generation by Reachability Analysis of SWFA *We adopt Linear Temporal Logic (LTL) (Kesten, Pnueli, and Raviv 1998) to define symbolic adversarial sequence on SWFA, which is shown below:*

$$\gamma \vDash \Diamond Z \tag{6}$$

*where γ denotes the intermediate state of SWFA, $\vDash$ denotes "satisfies", and $\Diamond$ denotes "eventually", Z denotes **000-***

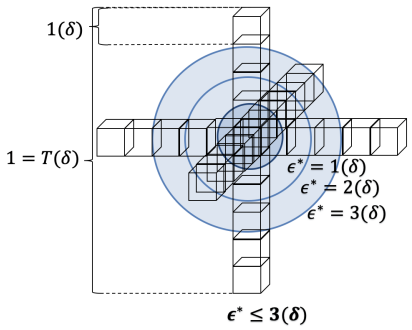

Figure 4: Abstract perturbation on symbolic blocks

status. *If γ eventually meets* ***000-status****, it means that it is a symbolic adversarial sequence.*

Finally, we use SWFA to rapidly check whether the abstract perturbated sample x_{pert} is an adversarial sequence. If *verifyBySWFA* returns *true*, we add it to $x^{*'}$, the list of symbolic adversarial sequences. We then use sampling techniques to instantiate symbolic adversarial sequences $x^{*'}$ into the set of concrete adversarial sequences x'. we assume that symbolic adversarial sequences generated by our approach are largely successful, so it is feasible to use arbitrary sampling methods, and we use random sampling here.

The novelty of our approach is that we conduct abstract perturbations on the symbolic space. It has two advantages. One is that the abstract perturbation has fewer possible iteration space, and the other is that we can generate concrete adversarial sequences from symbolic adversarial sequences in the batch, which brings more efficiency.

Experimental Evaluation

To demonstrate the effectiveness and efficiency of our approach, we conduct two experiments on well-trained LSTM from time-series data. The first is the implementation of the RNN-SWFA extraction algorithm using Fast k-DCP. The second is the implementation of SWFA-based adversarial sequence generation named AbASG. We compare it with FGSM (Goodfellow, Shlens, and Szegedy 2014), PGD (Madry et al. 2018), and NewtonFool (Jang, Wu, and Jha 2017). All our experiments are conducted on ubuntu20.0.4 which comes with one Intel Xeon Silver 4210R CPU and one GeForce RTX 3090 GPU. And all experiments are implemented in Pytorch with version1.7.1 within Python3.8.2. The datasets and the code in the experiments are available on the Github repository: https://github.com/AbASG/AbASG.

Datasets and Experimental Settings

UCR time-series datasets (Chen et al. 2015) and NGSIM (Montanino and Punzo 2013) are selected for our experiment. UCR datasets used in our experiment include ProximalPhalanxOutlineAgeGroup (PPOAG), Chinatown (CT), Earthquakes (EQ). To further demonstrate the effectiveness of our approach, we use a scenario programming language of autonomous driving, Scenic (Fremont et al. 2020), combined with the car simulator, Carla (Dosovitskiy et al. 2017), to generate the Autonomous Driving Datasets (ADD) with the sample size of 30,000. There are three classes of vehicle behavior, turning left, going straight, turning right, denoted as label y. It contains several features. Detailed descriptions of these datasets can be found in the Appendix.

We train a 2-hidden-layer LSTM for our experiment, and the number of hidden layer nodes is set to 1/200 of the datasets' size. The loss function is chosen as CrossEntropyLoss, and Adam is our choice for the optimizer. More detailed parameter settings of LSTM can also be found in the Appendix.

Benchmarks and Metrics

Five indicators are proposed for evaluating the SWFA's extraction: Accuracy of RNN (*AoR*), Accuracy of SWFA (*AoS*), Extraction Time (*ET*), RNN's Running Time (*RT*), SWFA's Running Time (*ST*). In *AoR* and *AoW*, the left value denotes the accuracy of training data, while the right value is that on test data. We use *RT* and *ST* for comparing the computation speed between RNN and SWFA on the total dataset.

And we use Attack Success Rate *(ASR)* and time consuming to evaluate the effect of our adversarial sequence generation algorithm AbASG. The perturbation unit of different attack algorithms are unified as δ. In FGSM and PGD, $\delta = 0.006$. In NewtonFool, $\delta = 0.01$.

Experiment I

We implement the extraction algorithm based on Fast k-DCP to extract SWFA from RNN on five different time-series datasets, and use the five indicators proposed above to evaluate them.

The reason why we design this experiment is that there may be some questions about our abstraction algorithm like: **RQ1:** Whether the extracted SWFA can really be applied in the infinite input? **RQ2:** What is the accuracy of the extracted SWFA? And what about the efficiency (the time of extraction and prediction)? **RQ3:** What kind of tasks are Fast k-DCP good at? We try to answer them through comparing and analysis of the experiment results.

Table 1: Comparison between SWFAs extracted by Fast k-DCP on various time-series data

Datasets	AoR(%)	AoS(%)	ET(s)	RT(s)	ST(s)
ADD	99/97	**89/79**	421.536	4.982	3.214
NGSIM	91/86	**77/73**	28.704	3.016	2.971
PPOAG	75/88	35/43	2.667	0.224	0.443
CT	53/74	53/73	0.026	0.005	0.004
EQ	82/75	**82/76**	7.229	1.112	1.194

Answer1: Just like what Table 1 illustrates, thanks to the symbolic representation of the input, SWFA can work for the infinite alphabet, as well as the test data, even if our algorithm has never seen them before. **Answer2:** As can be seen from the *AoS* column in Table 1, SWFA achieves high accuracy on both training and test data of ADD, NGSIM and EQ datasets, while its performance on PPOAG and CT

Table 2: Comparison between abstraction-based adversarial sequence generation approach and other adversarial attacking algorithms on the autonomous driving dataset

Category	**White Box**						**Black Box**			**Our Approach**		
Methods	**FGSM**			**PGD**			**NewtonFool**			**AbASG**		
Perturbation(δ)	1	5	10	1	5	10	1	2	3	1	2	3
ASR(%)	0.00	0.33	21.66	0.00	0.33	3.8	11.33	17.66	25.23	**20.52**	**34.36**	**53.72**
Time(s)	-	3.15	10.00	-	10.68	22.58	42.25	26.85	26.7	**39.94**	**20.42**	**18.55**

datasets is relatively not good enough since the accuracy of the original RNNs in these tasks are only 75% and 53% respectively. The phenomenon confirms that the accuracy of original RNN will definitely impact the performance of SWFA. However, we do not make any effort to improve RNNs since our algorithm is a general approach and does not rely on specified RNN. When it comes to efficiency, the running time of SWFA is almost the same as that of the original RNN on each dataset. Extraction time is validated to be the most time-consuming part, but its results can be easily reused. **Answer3:** For one thing, as mentioned above, the original RNN is expected to be well trained. For another, our algorithm is especially suitable for the tasks whose classification category and input dimension are relatively low.

Experiment II

We compare our adversarial sequence generation approach *AbASG* with several baseline methods. The white-box side includes FGSM (Goodfellow, Shlens, and Szegedy 2014) and PGD (Madry et al. 2018). The black-box side: NewtonFool(Jang, Wu, and Jha 2017).

We attempt to find the answers for the following questions about our adversarial sequence generation algorithm. **RQ4:** How efficient our adversarial sequences algorithm is and how can we prove it? **RQ5:** Are there any relation between the accuracy of SWFA and the efficiency of our adversarial sequence generation algorithm?

To answer the questions above, we conduct adversarial perturbation and instantiation with SWFA on the autonomous driving dataset, comparing it with traditional adversarial attacking algorithms. As shown in Table 2, we record the *Attack Success Rate (ASR)* and the time cost of different algorithms at different perturbation intensities. The perturbation unit of different algorithms is unified as δ.

Average Generation Time In particular, when we compare the generation time of adversarial examples, as shown in Table 2, we did not take extra consideration of SWFA extraction time. The reason is as follows:

$$T_{average} = \frac{\lambda_{ext} + n \cdot x}{n} = \frac{\lambda_{ext}}{n} + x \tag{7}$$

λ_{ext} denotes the SWFA extraction time, $T_{average}$ denotes the average time to find a single adversarial example, x denotes generation time of single adversarial example, n denotes the number of adversarial examples generated. We have the property that $n \to \infty$, $T_{average} \to x$, which means that when n becomes larger, the cost of building SWFA will be diluted until it approaches zero.

Finally, we can get the following answers from the experiment. **Answer4:** We use the *ASR* and generation times to verify the efficiency of our algorithm on the autonomous driving dataset. The former illustrates that our approach has an outstanding success rate, which respectively has 81.11%, 94.56% and 112.92% improvements compared with NewtonFool and is much better than the white-box attacking methods, despite the small perturbation ranges makes it even more difficult to achieve. The latter shows that our approach is 1.44 times faster than NewtonFool at most with the same perturbation and an even higher level of *ASR*. Not to mention the FGSM and PGD who are no longer fairly comparable in generation times to our algorithm because of the lower performance in this case. **Answer5:** Since SWFA is an abstract expression of RNN, if the accuracy of RNN is low, the guidance ability of extracted SWFA for perturbations will be limited, which will eventually reduce the efficiency of adversarial sequence generation. And the reverse is also true.

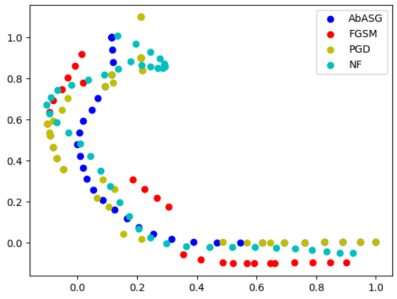

Figure 5: Adversarial sequence generated by different algorithms on Autonomous Driving Dataset

As shown in Fig 5, we use diverse algorithms to generate adversarial sequences on ADD. These colored point sets constitute adversarial trajectories. Compared with other algorithms, the adversarial trajectory generated by our approach has lower perturbation, which is harder for people to recognize it. The adversarial trajectories generated by other algorithms are not in line with the characteristics of vehicle trajectories. Therefore, our approach can achieve a better attack success rate under the condition of small perturbation.

Concluding Remarks

In this paper, we have proposed an efficient adversarial sequence generation approach for RNN by SWFA abstraction. The approach is applicable to generate covert adversarial sequences with high efficiency. We implemented the approach by the Fast k-DCP symbolic extraction algorithm reducing the complexity of perturbation. The main advantage of our approach is that the symbolic extraction technique constructs the abstract model of RNN. Besides, we set parameters to keep the perturbation as much as possible within the human-invisible range. Compared with the baseline of the adversarial example generation algorithms, our approach is more efficient with the spatio-temporal sequential dataset.
For future work, we would further optimize our approach, adapting to large-class sequential data. Multi-label classification tasks with too many categories may lead to lower accuracy of SWFA, which impacts the efficiency of adversarial sequence generation. Moreover, we will explore more detailed comparisons with adversarial example generation algorithms on various datasets and investigate the reachability analysis of SWFA.

Acknowledgements

Financial support for this work, provided by the National Natural Science Foundation of China under Grant No. 61972153, the Key projects of the Ministry of Science and Technology under No. 2020AAA0107800, is gratefully acknowledged.

References

Ayache, S.; Eyraud, R.; and Goudian, N. 2018. Explaining Black Boxes on Sequential Data using Weighted Automata. In Unold, O.; Dyrka, W.; and Wieczorek, W., eds., *Proceedings of the 14th International Conference on Grammatical Inference, ICGI 2018, Wrocław, Poland, September 5-7, 2018*, volume 93 of *Proceedings of Machine Learning Research*, 81–103. PMLR.

Chen, P. Y.; Zhang, H.; Sharma, Y.; Yi, J.; and Hsieh, C. J. 2017. ZOO: Zeroth Order Optimization based Black-box Attacks to Deep Neural Networks without Training Substitute Models. *ACM*.

Chen, Y.; Keogh, E.; Hu, B.; Begum, N.; Bagnall, A.; Mueen, A.; and Batista, G. 2015. The UCR Time Series Classification Archive.

Clarke, E. M.; Henzinger, T. A.; Veith, H.; and Bloem, R., eds. 2018. *Handbook of Model Checking*. Springer. ISBN 978-3-319-10574-1.

Dong, G.; Wang, J.; Sun, J.; Zhang, Y.; Wang, X.; Dai, T.; and Dong, J. S. 2019. Analyzing Recurrent Neural Network by Probabilistic Abstraction. *CoRR*, abs/1909.10023.

Dosovitskiy, A.; Ros, G.; Codevilla, F.; López, A. M.; and Koltun, V. 2017. CARLA: An Open Urban Driving Simulator. In *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*, volume 78 of *Proceedings of Machine Learning Research*, 1–16. PMLR.

Du, X.; Xie, X.; Li, Y.; Ma, L.; Liu, Y.; and Zhao, J. 2019. A Quantitative Analysis Framework for Recurrent Neural Network. In *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*, 1062–1065. IEEE.

Fremont, D. J.; Kim, E.; Dreossi, T.; Ghosh, S.; Yue, X.; Sangiovanni-Vincentelli, A. L.; and Seshia, S. A. 2020. Scenic: A Language for Scenario Specification and Data Generation. *CoRR*, abs/2010.06580.

Goodfellow, I.; Shlens, J.; and Szegedy, C. 2014. Explaining and Harnessing Adversarial Examples. *arXiv 1412.6572*.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. In *ICML*.

Gopinath, D.; Katz, G.; Pasareanu, C. S.; and Barrett, C. W. 2017. DeepSafe: A Data-driven Approach for Checking Adversarial Robustness in Neural Networks. *CoRR*, abs/1710.00486.

Gorrieri, R. 2017. *Labeled Transition Systems*, 15–34. Cham: Springer International Publishing. ISBN 978-3-319-55559-1.

Huang, X.; Kroening, D.; Ruan, W.; Sharp, J.; Sun, Y.; Thamo, E.; Wu, M.; and Yi, X. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Comput. Sci. Rev.*, 37: 100270.

Jang, U.; Wu, X.; and Jha, S. 2017. Objective Metrics and Gradient Descent Algorithms for Adversarial Examples in Machine Learning. In *Proceedings of the 33rd Annual Computer Security Applications Conference, Orlando, FL, USA, December 4-8, 2017*, 262–277. ACM.

Kesten, Y.; Pnueli, A.; and Raviv, L. O. 1998. Algorithmic verification of linear temporal logic specifications. In *Lecture Notes in Computer Science*.

Kurakin, A.; Goodfellow, I.; and Bengio, S. 2016. Adversarial examples in the physical world.

Li, Z.; Gu, Z.; Di, X.; and Shi, R. 2020. An LSTM-Based Autonomous Driving Model Using Waymo Open Dataset. *CoRR*, abs/2002.05878.

Madry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards Deep Learning Models Resistant to Adversarial Attacks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Michalenko, J. J.; Shah, A.; Verma, A.; Baraniuk, R. G.; Chaudhuri, S.; and Patel, A. B. 2019. Representing Formal Languages: A Comparison Between Finite Automata and Recurrent Neural Networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Montanino, M.; and Punzo, V. 2013. Making NGSIM Data Usable for Studies on Traffic Flow Theory: Multistep Method for Vehicle Trajectory Reconstruction. *Transportation Research Record*, 2390(1): 99–111.

Omlin, C. W.; Thornber, K. K.; and Giles, C. L. 1996. Representation of fuzzy finite state automata in continuous recurrent, neural networks. In *Proceedings of International Conference on Neural Networks (ICNN'96), Washington, DC, USA, June 3-6, 1996*, 1023–1027. IEEE.

Suzuki, K.; Hendrian, D.; Yoshinaka, R.; and Shinohara, A. 2021. Query Learning Algorithm for Symbolic Weighted Finite Automata. In Chandlee, J.; Eyraud, R.; Heinz, J.; Jardine, A.; and van Zaanen, M., eds., *Proceedings of the Fifteenth International Conference on Grammatical Inference*, volume 153 of *Proceedings of Machine Learning Research*, 202–216. PMLR.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013a. Intriguing properties of neural networks.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013b. Intriguing properties of neural networks. *Computer Science*.

Wei, X.; Guo, Y.; and Li, B. 2021. Black-box adversarial attacks by manipulating image attributes. *Inf. Sci.*, 550: 285–296.

Weiss, G.; Goldberg, Y.; and Yahav, E. 2018. Extracting Automata from Recurrent Neural Networks Using Queries and Counterexamples. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, 5244–5253. PMLR.

Zhang, X.; Du, X.; Xie, X.; Ma, L.; Liu, Y.; and Sun, M. 2021. Decision-Guided Weighted Automata Extraction from Recurrent Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13): 11699–11707.

Zhou, X.; Guo, Y.; Shen, M.; and Yang, G. 2020. Artificial Intelligence in Surgery. *CoRR*, abs/2001.00627.