TCN-ATT: A Non-recurrent Model for Sequence-Based Malware Detection

Junyao Huang[1], Chenhui Lu[1], Guolou Ping[1], Lin Sun[1], and Xiaojun Ye[1,2](✉)

[1] School of Software, Tsinghua University, Beijing, China
{junyao-h17,luch18,pgl19}@mails.tsinghua.edu.cn,
yexj@mail.tsinghua.edu.cn

[2] National Engineering Laboratory for Big Data System Software, Tsinghua University, Beijing, China

Abstract. Malware detection based on API call sequences is widely used for the ability to model program behaviours. But RNN-based models for this task usually have bottlenecks in efficiency and accuracy due to their recurrent structure. In this paper, we propose a Temporal Convolutional Network with ATTention (TCN-ATT) architecture, which processes sequences with high parallelization and is robust to sequence length. The proposed TCN-ATT consists of three components: (1) a TCN module which processes sequence with convolutional structure, (2) an attention layer to select effective features and (3) a split-and-combine mechanism to fit inputs with various size. A formalized deduplication method is also proposed to reduce redundancy with less information loss. According to our experiments, the proposed model reaches an accuracy of 98.60% and reduces time cost by over 60% compared with existing RNN-based models.

Keywords: Malware detection · API call sequence · Deep neural network · Temporal Convolutional Network · Attention mechanism

1 Introduction

In recent years, more and more entities are storing their valuable information in places reachable through networks, which in some way makes themselves potential victims of malicious applications (malwares). Malware attacks have also increased greatly in both quantities and categories. Malware detectors based on signature database [1,6,17] or static analysis [14] are faced with increasing difficulty because they are often vulnerable to obfuscation methods [11]. So many researchers put effort into dynamic analysis and develop algorithms to identify malicious programs through their behaviors. In dynamic analyses, system API call sequences are most frequently used to represent the behaviors of programs. Data mining and traditional machine learning methods are often employed to handle malware detection tasks based on API call sequences [7,15,19]. These methods usually require low dimensional statistical features as the input and thus expertise-based feature engineering is necessary. These requirements lead to a bottleneck in accuracy.

With the development of deep learning models, many new models have been proposed to detect malware based on raw API call sequences and most of them are using recurrent neural network (RNN) models [12,13,16,18]. These RNN-based models reach better accuracy than data mining and traditional machine learning methods but challenges still exist. The recurrent architecture causes inevitable low parallelization when processing and brings uncertainty in receptive field size on input sequences. Furthermore, recently published models become increasingly complicated and are combined with various other analysis methods. In this situation, we propose a Temporal Convolutional Network with ATTention (TCN-ATT) architecture, which is a relatively simple and effective non-recurrent model, to detect malwares based on API call sequences. Noticing that the length of API call sequences varies from 100 to 20000, these whole sequences are not proper inputs for either recurrent or convolutional models. So a split-and-combine method is proposed in the TCN-ATT model.

Our contributions are:

– For the first time, TCN is introduced to malware detection based on API call sequences (Sect. 3.2) bringing considerable accuracy and high efficiency. To further improve the accuracy, a specifically designed attention layer is employed in our architecture (Sect. 3.3).
– We propose a sequence splitting method together with a corresponding loss function for the detection task in view of API call sequences characteristics (Sects. 3.1 and 3.4). They control the model size and hold the accuracy no matter how the length of input sequences varies. They can also be applied to other models on sequence-based malware detection task.
– We give a deduplication method to improve the performance of our model as well as other sequence-based models (Sect. 2.2). We define two parameters for the method to control the deduplication intensity. This method helps to reduce redundancy while retaining some repetition behavior information.

2 Data Preprocessing

Before introducing the TCN-ATT model, we give a brief introduction about procedures that are implemented in front of the model, including the content of input sequences and the proposed deduplication method. Firstly, API call sequences are extracted from programs by running them in virtual environments. Then the sequence data go through some preprocessing to make them fit for the model. Finally, these preprocessed data are fed to TCN-ATT model, which will give a decision about whether the program is benign or malicious.

2.1 Malware Behavior Representation

For each executable file, an API call sequence is extracted from sandbox to represent the behavior and the function name of each API call is used. Each API function is represented by a specific integer which is finally transformed to a one-hot vector in training and testing steps.

2.2 Duplicate API Sequences Processing

After analyzing API call sequences, we find out that one API is often called multiple times consecutively and this kind of repetition also happens to some subsequences consisting of several APIs. This happens to both benign and malicious softwares, because the program sometimes do some similar tasks consecutively. In order to reduce the length of sequences fed to the model, Kolosnjaji et al. [10] and Xiaofeng et al. [18] mentioned some methods to remove continuous same API functions in sequences. [10] dose not consider the case of repetition of an API group and [18] simply removes all the duplicates. In this paper we propose a deduplication method which is similar to theirs but more flexible.

In our consideration, the duplicates of an API call subsequence pattern should be reduced to avoid information redundancy but should not be totally removed because the repetition itself contains program behavior information. So, we define two parameters for the duplicate reducing method: l_m, the max length of a target pattern; k, the max number of consecutive duplicates kept for a pattern. For example, given a sequence $A_1A_2A_3A_1A_2A_3A_1A_2A_3$: when we set $l_m = 3$ and $k = 2$, the de-duplicated sequence is $A_1A_2A_3A_1A_2A_3$; when we set $l_m = 2$ and $k = 2$, $A_1A_2A_3$ is not regarded as a pattern with $len(A_1A_2A_3) = 3 > l_m$ and therefore this sequence stays unchanged after deduplication.

As described above, this design removes less valuable duplicates and keeps some repeating behavior information. This deduplication method is proved to bring accuracy improvement for models, according to experiments in Sect. 4.2.

3 TCN-ATT Model

As shown in Fig. 1, the whole preprocessed sequence from each sample is firstly split into several subsequences with a fixed size. Each subsequence is fed into a network containing the TCN module [2], an attention layer and a fully connected (FC) layer. Then the network produces a sub-prediction for the subsequence. Finally, these sub-predictions are fed into a task-specific loss function to train the whole model in the training phase and analyzed by specific rules to give a sample-level prediction in the practicing phase.

3.1 Sequence Splitting

The length of dynamic extracted API call sequences is usually rather big and varies a lot among different samples, even after deduplication preprocessing. For both the TCN-ATT model and recurrent models, it does not lead to good results to feed each sequence into the model without splitting it. So we split each sequence into parts of a fixed size. Thus the input of the TCN model will be the subsequences instead of the whole API call sequence. Sequence splitting also brings a benefit that the size of the model only depends on the subsequence length setting regardless of whole sequence size. A proper subsequence length n for the model is chosen by experiments, which will be described in Sect. 4.4.

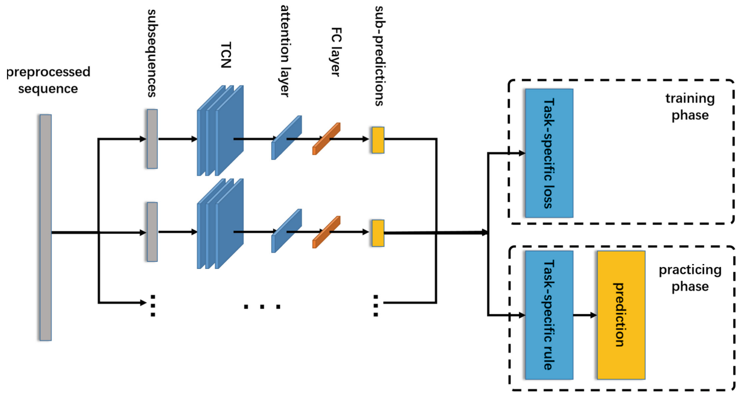

Fig. 1. Process of each sample's sequence in TCN-ATT model

Although this method solves the issue of sequence length variation, it brings some difficulties in combining partial results into a sample-level classification conclusion. Therefore, a task-specific loss for training phase and a result-combining method for predicting phase are designed to cooperate with splitting method, which will be described in Sect. 3.4.

3.2 Temporal Convolutional Network

RNN models have a bottleneck in accuracy and efficiency when faced with mass data due to its recurrent structure. So we introduce a non-recurrent model to take the place of RNNs.

Temporal Convolutional Network (TCN) is a network architecture proposed by Bai et al. [2]. This fully convolutional network produces an output of the same length as the input, similar to RNNs. With the utilization of *dilated convolutions* and *residual connections*, the TCN is able to allow very long effective history with a rather shallow network. And it is worth mentioning that the ability is of great importance to a malware detector based on API call sequences.

A simple convolution only has an input reception field with size linear in the depth of the network. This makes it challenging to apply it to sequence tasks, especially those requiring longer history. So *dilated convolutions* are employed to enable a larger receptive field with size exponential in the depth of the network. To put it formally, for a 1-D sequence input $\boldsymbol{x} \in \mathbb{R}^n$ and a convolutional filter $f : \{0, ..., k-1\} \rightarrow \mathbb{R}$, the dilated convolution operation F on elements end with index s of the sequence is defined as:

$$F(s) = (\boldsymbol{x} *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \boldsymbol{x}_{s-d \cdot i} \tag{1}$$

where d is the dilation factor, k is the filter size, and each $(s - d \cdot i)$ is index of an element from the 'past' part in the input $\boldsymbol{x}$. Using this type of convolution, the effective history of one such layer is $(k-1)d$. Furthermore, d is increased exponentially with the depth of the network (i.e., $d = O(2^i)$ at level i).

With above designs, the TCN model is able to take similar inputs and produce similar outputs as RNNs while it is efficient taking advantage of convolution architectures. In our architecture, we use the sequence-to-sequence (seq2seq) mode of TCN. To be specific, it works as described below.

For a 1-D sequence input $\boldsymbol{x} \in \mathbb{R}^n$ representing an API call sequence, $\boldsymbol{x}'$ is its corresponding one-hot-encoded form with a size of $n \times m$, where n is the length of the sequence and m is the total number of involved API functions. Then $\boldsymbol{x}'$ is fed into the TCN and the module finally produce an output $\boldsymbol{H}$ with a size of $n \times c$, where c is the size of the output feature that TCN produces for each time of the sequence. We choose this mode because we expect the network to produce more suggestive results and to have more interpretabilities with the attention layer described in the next subsection.

3.3 Attention Layer

Models with attention mechanism are now the state-of-art for multiple tasks [3]. Attention mechanism is usually able to improve the performance as well as the interpretability of various models. In the TCN-ATT model, we also design an attention layer between the TCN and the FC layer. This attention layer helps to reduce the size of the feature matrix produced from TCN while keeps important information in it and therefore improves the model performance. For the aforementioned output $\boldsymbol{H} \in \mathbb{R}^{n \times c}$, the operation of the attention layer is:

$$\boldsymbol{\alpha} = softmax\left(tanh(\boldsymbol{H})\boldsymbol{\mu}^T\right) \tag{2}$$

$$\boldsymbol{w} = \boldsymbol{\alpha}^T \boldsymbol{H} \tag{3}$$

where $\boldsymbol{\mu} \in \mathbb{R}^{1 \times c}$ is the attention factor that we expect the model to learn, and $\boldsymbol{w} \in \mathbb{R}^{1 \times c}$ is the final vector that represents the feature of the input $\boldsymbol{x}$. We can see $\boldsymbol{\alpha} \in \mathbb{R}^{n \times 1}$ as a coefficient vector calculated from $\boldsymbol{H}$ and $\boldsymbol{\mu}$ representing the importance of feature vectors produced from all the n time steps.

Through this attention layer, the original feature $\boldsymbol{H}$ with size $n \times c$ is compressed to the final feature $\boldsymbol{w}$ with size of $1 \times c$. Furthermore, we can analyze the importance of subsequences for an input $\boldsymbol{x}$ via the calculated $\boldsymbol{\alpha}$.

3.4 Task-Specific Loss

Each subsequence $\boldsymbol{x}$ with fixed length n is transformed to a feature vector $\boldsymbol{w} \in \mathbb{R}^{1 \times c}$ through the TCN model and the attention layer. Then $\boldsymbol{w}$ is fed into a FC layer and a softmax layer to produce a prediction result $\boldsymbol{p} \in \mathbb{R}^2$, and $\hat{y} = argmax(p)$ is the label that the model predicts (0 as benign and 1 as malicious). For each sequence sample, which is split into a set of subsequence $\boldsymbol{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_k\}$, the model will give a set of predictions $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_k\}$ and $\hat{Y} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_k\}$. While in other classification tasks the model is usually expected to make all sub-predictions of one sample close to the ground truth label, it works differently in this task.

One malware may not run maliciously all the time and the API call sequences extracted can also have benign parts. Thus, the part-combining procedure on this task is different. In the predicting phase, we regard a sample with all subsequences predicted to 0 as benign, while regarding a sample with at least one subsequence predicted to 1 as malicious. Under this consideration, we should avoid pushing the model too hard and allow the model to produce some benign sub-predictions for malicious samples. Therefore, we give a task-specific loss function $L(y, \boldsymbol{P})$ to calculate the total loss of a prediction set $\boldsymbol{P} = \{\boldsymbol{p}_1, \boldsymbol{p}_2, ..., \boldsymbol{p}_k\}$ from a sample s:

$$\begin{aligned} L(y, \boldsymbol{P}) &= \chi_{y=0|\max \hat{Y}=0} \sum_{i=0}^{k} L(y, \boldsymbol{p}_i) \\ &\quad + \left(1 - \chi_{y=0|\max \hat{Y}=0}\right) \sum_{i=0}^{k} \left(\left(1 - \beta \cdot \chi_{\hat{y}=0}\right) L(y, \boldsymbol{p}_i)\right) \end{aligned} \tag{4}$$

$$L(y, \boldsymbol{p}_i) = -\left(y \log \boldsymbol{p}_{i,0} + (1-y) \log \boldsymbol{p}_{i,1}\right) \tag{5}$$

$$\hat{y} = argmax(p) \tag{6}$$

$$\hat{Y} = \{\hat{y}_1, \hat{y}_2, ..., \hat{y}_k\} \tag{7}$$

where y is the ground truth label of the sample, $\hat{y}_i$ is the predicted label for each subsequence, and $\beta \in [0, 1]$ is a hyper-parameter designed to reduce the loss from specific subsequences. χ is an indicator function. χ_g equals 1 when g is true and 0 otherwise. To put it simply, $L(y, \boldsymbol{P})$ equals $\sum_{i=0}^{k} L(y, \boldsymbol{p}_i)$ when the sample is labeled benign or when the model predicts no subsequences of a malicious sample to be malicious. When there are some subsequences predicted to be malicious in a malicious sample, the hyper-parameter β will restrict the loss function to reduce the punishment on benign sub-predictions, since they may be actually correct.

4 Experiments

In this section, we evaluate the effectiveness of TCN-ATT model on the malware detection task as well as the influence of the proposed designs and settings, including API sequence deduplication method, the attention layer itself and hyper-parameters. The efficiency of the proposed model is also evaluated.

4.1 Dataset and Evaluation Metrics

We collect over 6900 malicious PE files from Malekal website[1], CILPKU08 dataset and Henchiri-Dataset[2]. Also, over 2700 benign PE files are acquired

[1] http://www.malekal.com.
[2] http://www.cil.pku.edu.cn/resources/.

from Windows system files or downloaded from several websites (e.g. completely free software, softonic). We then check these files by uploading them to the VirusTotal[3] website in order to keep some mislabeled samples out of the dataset. According to VirusTotal results, the final dataset contains malware from families including Backdoor, Trojan-Downloader, Trojan-Ransom, AdWare and Worm. In general, our dataset contains 2497 malicious samples and 2497 benign samples. We use 5-fold cross validation to evaluate different methods. Thus at each time, 80% samples are used for training and 20% are for testing.

To evaluate the performance of different mechanisms, the following evaluation metrics are used: accuracy (ACC), precision (PR), recall (RC), receiver operating characteristic (ROC) curve and area under curve (AUC).

These files are run in Cuckoo sandbox, which can extract API call sequences while files are running in a Windows 7 virtual environment. After being preprocessed as descried in Sect. 2, these sequences are fed into TCN-ATT model.

We implement the TCN-ATT model and other models envolved in experiments by python 3.6.5 with Tensorflow and Scikit-Learn. We train and test these models in a Ubuntu system with 8 GTX-1080Ti GPUs.

4.2 Effect of Deduplication

We first conduct an experiment to evaluate the effect of the proposed deduplication method. Original sequences and shortened sequences converted from original ones are used to train several models respectively. Then we evaluate trained models by accuracy. We choose RNN, LSTM from RNN family, LSTM with the attention layer, as well as TCN and our TCN-ATT model. The setting $\{l_m = 5, k = 2\}$ are proved to work best in pre-experiments and thus only results under this setting are shown in Table 1 for simplicity. From Table 1 we can see an evident increase in the ACC of each model fed with shortened sequences, which indicates that the proposed deduplication method is effective in improving the performance of a diverse range of models in this task. So we use shortened sequences as the input in all the following experiments.

Table 1. Accuracy using different input sequences

Model	ACC(%)	
	Original sequence	Reduced sequence
RNN	92.02	93.84
LSTM	94.23	95.19
LSTM+attention	96.46	97.59
TCN	96.69	97.78
TCN-ATT	97.59	98.60

[3] https://www.virustotal.com/.

4.3 Malware Detection

In this part, we compare the TCN-ATT model with traditional machine learning methods and some deep learning models:

- **Decision Tree/Naive Bayes/SVM/Random Forest**. Popular traditional machine learning methods. Directly feeding sequences or subsequences into these models leads to poor results. So a transition probability matrix is calculated as the feature vector for each sample.
- **RNN/LSTM/GRU.** Widely-used recurrent models in sequence tasks [4,8] under split-and-combine method without attention layer.
- **TCN.** The original TCN model under split-and-combine method without the attention layer.
- **LSTM/GRU with attention.** LSTM/GRU model under split-and-combine method with the attention layer.
- **GRU attention and TCN attention without sequence splitting.** Models of this group are fed with whole sequences and they make predictions directly with no combining method.
- **CNN+LSTM.** A model containing two convolutional layers and one LSTM layer proposed by Kolosnjaji et al. [10].
- **Bi-Residual LSTM.** A LSTM-based model containing two bidirectional layers with residual connection proposed by Xiaofeng et al. [18].
- **TCN-ATT.** The proposed model in this paper.

The dropout rate is 0.5 and single feature size (c in Sect. 3) is 128 for RNNs/LSTMs/GRUs/TCNs envolved. For models under split-and-combine method, the input sequences are split with window size 600 and β in loss the

Table 2. Accuracy of different models in malware detection

No	Model	PR(%)	RC(%)	ACC(%)
1	Decision Tree	77.99	93.50	83.68
2	Naive Bayes	67.83	77.60	70.37
3	SVM	83.13	93.60	87.28
4	Random Forest	90.62	92.21	91.29
5	RNN	94.45	93.30	93.84
6	LSTM	93.40	96.94	95.19
7	GRU	96.59	97.37	96.98
8	TCN	96.67	98.99	97.78
10	GRU+attention (no split)	95.98	96.16	96.05
11	TCN+attention (no split)	96.90	97.13	97.01
12	LSTM+attention	97.41	97.78	97.59
13	GRU+attention	97.35	98.73	98.04
14	CNN+LSTM [10]	92.76	93.55	93.15
15	Bi-Residual LSTM [18]	96.14	96.59	96.26
16	TCN-ATT(ours)	**98.02**	**99.18**	**98.60**

function is 0.25. For non-attention deep models, all the $\boldsymbol{H} \in \mathbb{R}^{n \times c}$ is fed into the FC layer. The dilations setting is [1, 2, 4, 8, 16, 32] in all TCNs. Hyper-parameter values of other methods are also carefully selected to reach their best accuracy.

Table 2 shows detection accuracy, precision and recall of above models. Comparing models 1∼4 with models 5∼16, we can conclude that deep models perform better than traditional machine learning models when using API call sequences as the input. Results of models 5∼8 show build-in abilities of original models under the split-and-combine structure. The model containing TCN outperforms other three recurrent models. From models 6∼8 and models 12, 13, 16, it is observed that the attention layer brings a significant improvement to original models. This layer allows the model to selectively focus on important parts of the whole output feature and thus help obtain better results. Similarly, results of model 10, 11 and model 13, 16 indicate that our split-and-combine method brings considerable performance improvement to these models. From above results, we can draw a conclusion that the TCN-ATT model outperforms other models in Table 2, since it reaches the highest accuracy, precision and recall.

4.4 Hyper-parameters

We also conduct experiments to choose hyper-parameter values of the TCN-ATT model. We evaluate different settings by accuracy and AUC. The results are presented in Table 3 and Fig. 2.

Table 3. Effect of different hyper-parameters

Hyper-parameter	Value	ACC(%)	AUC
Window size	150	97.29	0.9901
	300	97.70	0.9914
	600	**98.60**	**0.9955**
	900	98.35	0.9925
	1200	97.92	0.9915
Dilations	[1, 2, 4, 8]	97.00	0.9857
	[1, 2, 4, 8, 16]	97.88	0.9938
	[1, 2, 4, 8, 16, 32]	**98.60**	**0.9954**
	[1, 2, 4, 8, 16, 32, 64]	98.29	0.9949
Dropout rate	0.0	98.19	0.9916
	0.25	98.40	0.9938
	0.5	**98.60**	**0.9954**
	0.75	98.40	0.9941
Overlap rate	0.75	98.29	0.9931
	0.5	98.29	0.9931
	0.25	98.45	0.9952
	0	**98.60**	**0.9955**

The window size (i.e. subsequence length) is an important hyper-parameter of the TCN-ATT model. With a bigger window size, the TCN module is able to take a longer subsequence as its input, which brings less sequence splitting but more training cost. As shown in Table 3, the accuracy and AUC do not always increase when the window size grows and 600 reaches the best result.

The dilations setting defines the parameter d in each dilated convolution layer (see Sect. 3.2) and the number of layers. It also highly affects the proposed model. As the dilations go deeper, the receptive field of the top cell becomes larger, while training becomes more difficult. Similar to the window size hyper-parameter, the best result comes from a middle value and we regard $dilations = [1, 2, 4, 8, 16, 32]$ as the trade-off between the receptive field and the training cost.

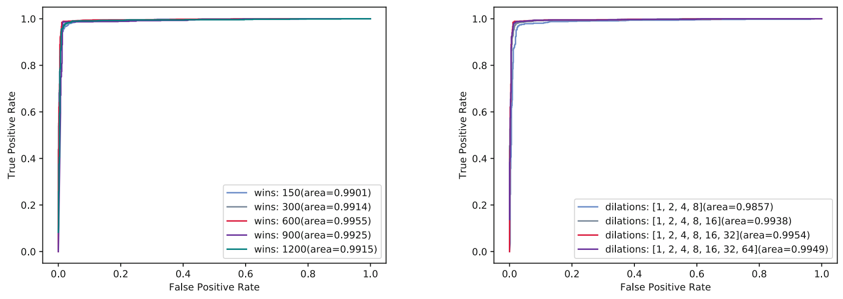

Fig. 2. ROC of some hyper-parameters

According to our experiments, the dropout rate seems to have little influence on the model. The accuracy is over 98.2% whichever value we choose from 0.0 to 0.75. And the best dropout rate is 0.5.

The overlap rate represents how much one subsequence overlaps its neighbors. We did not mention this setting in Sect. 3 because it is indicated that the best result is reached when there is no overlap.

For brevity, other hyper-parameters of less importance are not discussed here.

4.5 Efficiency

We also conduct experiments to evaluate the efficiency of TCN-ATT model. A set of 1000 samples is randomly selected from our dataset and is expanded to 10000 by repeating samples. Experiments are conducted in both training phase and testing phase on this dataset and the time cost of each model is recorded. This is performed on TCN-ATT model as well as RNN/LSTM/GRU with splitting and attention mechanism. In these experiments, only **one** GPU is used in case that TCN can run on multi-GPU mode while recurrent models are not able to. All models involved use settings that can reach the best accuracy as described in Sect. 4.3 and the batch size is set to 32. Table 4 shows the results.

Table 4. Time cost of some models

Model	Time cost (ms/sample)	
	Training phase	Testing phase
RNN+attention	16.4	8.0
LSTM+attention	20.1	9.2
GRU+attention	23.1	9.0
TCN-ATT	**8.8**	**3.6**

As expected, LSTM and GRU cost more time than the original RNN while reaching higher accuracy. However, taking advantage of convolutional architectures, TCN-ATT outperforms above three recurrent models in terms of time cost. It saves time by 62% in training and 60% in testing compared with the GRU+attention model. So it's indicated that the TCN-ATT model has not only high accuracy but also excellent efficiency.

5 Related Works

API call information is widely used in malware detection. Static analysis extracts API calls from portable execution files [5], log files [16] and DEX files on mobile platforms [9,19]. And API call sequences can be captured dynamically as well. Based on API call sequences, Ravi et al. [13] use Markov chain to model the sequences and designed a data mining algorithm to generate the classification rules. Some researchers apply machine learning methods for classification. Hansen et al. [7] utilize random forest algorithm to classify the malware based on API call sequences and API call frequency.

In recent years, the development of deep learning have greatly influenced malware detection methods. Pektas et al. [12] construct an API call graph and used graph embedding methods to generate graph embeddings. The normalized graph embeddings are forwarded into a deep neural network for classification. Since recurrent neural networks have good performance in tackling sequence data, Tobiyama et al. [16] use the RNN to extract feature vectors from the input API sequence, convert the feature vectors into images and apply a CNN to classify the images. Kolosnjaji et al. [10] process the API sequences via deep neural network which is composed of CNN and LSTM. Lu et al. [18] utilize the Bidirectional Residual LSTM to process the API sequence data and use machine learning methods based on API statistic features. Most of these sequence-based models rely on recurrent structures, which process long inputs sequentially and thus limit their performance.

6 Conclusion

In this paper, we present a convolutional network architecture called TCN-ATT for malware detection based on API call sequences. A temporal convolutional module and an attention layer are employed for stronger feature extraction ability. We also design a sequence splitting method and a task specific loss to enhance robustness for long sequences while controlling the model size. For sequence preprocessing, a formalized deduplication method with two parameters is proposed. It brings accuracy rise for our architecture and other sequence-based models. With above techniques, the proposed architecture obtains an accuracy of 98.60% and reduces time cost by over 60% compared with recurrent models. Experimental results indicate that the proposed approach is an effective classifier for automatic malware detection task. In the future, a sub-prediction combining method with more intelligence technique can be designed to bring more robustness and adaptability. Furthermore, analyses on attention layer values can be conducted to find out what the model focuses on and help to improve the performance.

Acknowledgement. This work is supported by National Key Research and Development Program of China (No. 2019QY1402).

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM **18**(6), 333–340 (1975)
2. Bai, S., Kolter, J.Z., Koltun, V.: An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. arXiv preprint arXiv:1803.01271 (2018)
3. Chaudhari, S., Polatkan, G., Ramanath, R., Mithal, V.: An attentive survey of attention models. arXiv preprint arXiv:1904.02874 (2019)
4. Cho, K., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734 (2014)
5. Fan, Y., Ye, Y., Chen, L.: Malicious sequential pattern mining for automatic malware detection. Expert Syst. Appl. **52**, 16–25 (2016)
6. Faruki, P., et al.: Android security: a survey of issues, malware penetration, and defenses. IEEE Commun. Surv. Tutorial. **17**(2), 998–1022 (2014)
7. Hansen, S.S., Larsen, T.M.T., Stevanovic, M., Pedersen, J.M.: An approach for detection and family classification of malware based on behavioral analysis. In: 2016 International Conference on Computing, Networking and Communications (ICNC), pp. 1–5. IEEE (2016)
8. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
9. Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D.: Maldozer: automatic framework for android malware detection using deep learning. Digital Invest. **24**, S48–S59 (2018)
10. Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C.: Deep learning for classification of malware system call sequences. In: Kang, B.H., Bai, Q. (eds.) AI 2016. LNCS (LNAI), vol. 9992, pp. 137–149. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-50127-7_11
11. Kuzurin, N., Shokurov, A., Varnovsky, N., Zakharov, V.: On the concept of software obfuscation in computer security. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 281–298. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75496-1_19

12. Pektaş, A., Acarman, T.: Deep learning for effective android malware detection using API call graph embeddings. Soft Comput. **24**(2), 1027–1043 (2020)
13. Ravi, C., Manoharan, R.: Malware detection using windows api sequence and machine learning. Int. J. Comput. Appl. **43**(17), 12–16 (2012)
14. Saxe, J., Berlin, K.: Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th International Conference on Malicious and Unwanted Software (MALWARE), pp. 11–20. IEEE (2015)
15. Shijo, P., Salim, A.: Integrated static and dynamic analysis for malware detection. Procedia Comput. Sci. **46**, 804–811 (2015)
16. Tobiyama, S., Yamaguchi, Y., Shimada, H., Ikuse, T., Yagi, T.: Malware detection with deep neural network using process behavior. In: 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), vol. 2, pp. 577–582. IEEE (2016)
17. Wu, S., Manber, U., et al.: A fast algorithm for multi-pattern searching. Technical report TR-94-17 (1994)
18. Xiaofeng, L., Fangshuo, J., Xiao, Z., Shengwei, Y., Jing, S., Lio, P.: ASSCA: API sequence and statistics features combined architecture for malware detection. Comput. Netw. **157**, 99–111 (2019)
19. Zhao, C., Zheng, W., Gong, L., Zhang, M., Wang, C.: Quick and accurate android malware detection based on sensitive APIs. In: 2018 IEEE International Conference on Smart Internet of Things (SmartIoT), pp. 143–148. IEEE (2018)