Using A Stacked-LSTM Architecture to Analyze API Call Sequences from Dynamic Malware Sandbox

Advay Balakrishnan[1] and Guilliermo Goldsztein[2#]

[1]Valley Christian High School
[2]Georgia Tech University
[#]Advisor

ABSTRACT

This paper explores the temporal hierarchy of API sequence calls using Deep Recurrent Neural Networks. RNN's have the capability to easily capture the nature of time series processing due to their sequential structure, whereas generic neural networks cannot find an underlying relationship between data that is separated by timesteps. The primary goal of the paper is to describe an approach to analyzing malware attacks on networks while accounting for each timestep of data and gaining more flexibility in the size of the detection request using the features of the Long Short-Term Memory (LSTM) architecture. We will also touch on the more recent Gated Recurrent Units (GRU) network architecture. We will discuss the tested network configurations and the most optimal model. Lastly, the paper will suggest some applications of this model and how it could be integrated into standard operating systems.

1. Introduction

Malwares are a growing concern amongst security professionals, and they are evolving rapidly: though cybercriminals previously targeted individuals with their malicious software, recently, even large corporations have become vulnerable to infections. In 2020, Mimecast, an IT security company, found that 51 percent of organizations experienced a malware attack, which led to disruptions in their business; in the following year, 61 percent of organizations reported ransomware attacks [1]. Malwares such as a Trojan (a software that is self-replicating) can cause data corruption, data theft, and even file deletions. As malwares become more complex, traditional malware analysis tools will also require increasing complexity, and new methods that are more adaptable, which can be achieved through a deep learning approach involving malware API sequences and a unique application of RNNs.

Neural networks have become popular in several fields because they can find relationships between a desired input vector and a corresponding output vector, a concept known as supervised learning. Due to its flexibility and feature engineering, the process of transforming data into a vectorized format, deep learning can be involved in object detection, language processing, suggestion algorithms, etc. The success of neural networks can be attributed to the ability to add hidden layers to models, increasing the depth of the network (hence the name "deep learning"), and in turn increasing the complexity of the model by additional parameters.

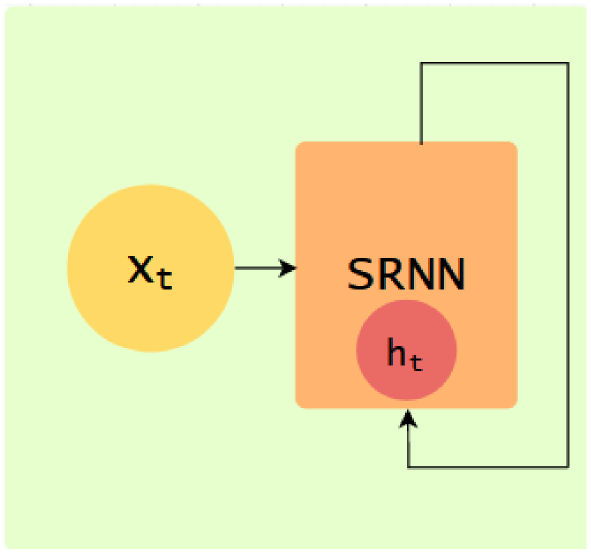
Recurrent Neural Networks changed how sequential data, or "series" data, was analyzed. Prior to RNNs, it was not effective to pass sequences into the typically neural network because of their inability to identify a connection between the timesteps in the data. RNNs made sequence problems such as sentiment classification of sequences of words possible through embedding words into a real vector domain, and passing these vectors into the model [2]. More importantly, RNNs don't have predetermined limits on size, granted the hidden state.

Although simple RNNs can learn the dependency between neighboring series data, they suffer from the vanishing gradient problem, or when dependencies in longer series cannot be retained. This inspired the Long Short-Term

Memory (LSTM) network; this network introduces gates that enable the model to select important information from the data sequences. This model also is effective at learning long term dependencies due to its connected cell state [3]. The Gated Recurrent Units (GRU) is another recurrent neural network model that is less complex than the LSTM because it has two gates, while an LSTM has three gates.

2. Further Review of Prior Work

As mentioned prior to this section, LSTMs have an advantage over simple RNN in that they have gates to store memory for longer periods of time through weights stored in the form of matrices and an activation function (sigmoid) to determine the relevance of a particular part of the sequence.



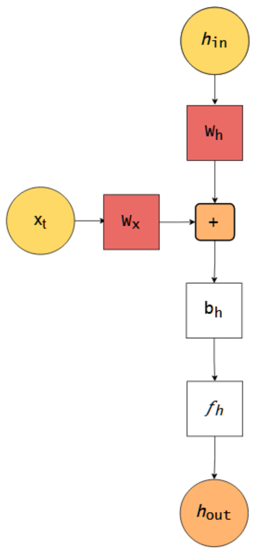


Figure 1. RNN Unit

Figure 2. Simple RNN Structure

Figure 1 shows the basic structure of a general RNN model. The model accepts an input that needs to be a series as a matrix of dimension three. This can be understood as a series of some integer **n** two-dimensional arrays, each containing some integer **t** dimensional vectors, which are of some integer size **k.** With this understanding, x_t in the model represents any two-dimensional array of size (t, k) in the dataset. Something important to note is the unique property of Recurrent Neural Networks in general that all of them contain a recurring unit. This is indicated by the arrow pointing back to the SRNN cell with a new input, h_t. This new value is a hyperparameter denoted as the hidden state, which is what the model gains from the specific timestep's data. This hidden state is passed into the SRNN cell each time a new timestep is processed, therefore, the model can be represented by multiple RNN Units such as the one above. Figure 2 depicts the mathematical properties of the simple recurrent neural network: x_t and h_{in} are the two inputs, and they produce a hidden state output. The weight for x acts as a matrix transformation to convert both vectors into the same dimension, which allows for their addition.

$$T: M_x \rightarrow M_h$$

The vector b_h is added to the final product through broadcasting, and an activation function is applied at the end of the computation. The result can be represented by: $h_{out} = f_h(h_{in}W_h + x_tW_x + b_h)$. This function is applied at every timestep of the sequence, and therefore it is repeated for all **t** vectors in the sample.

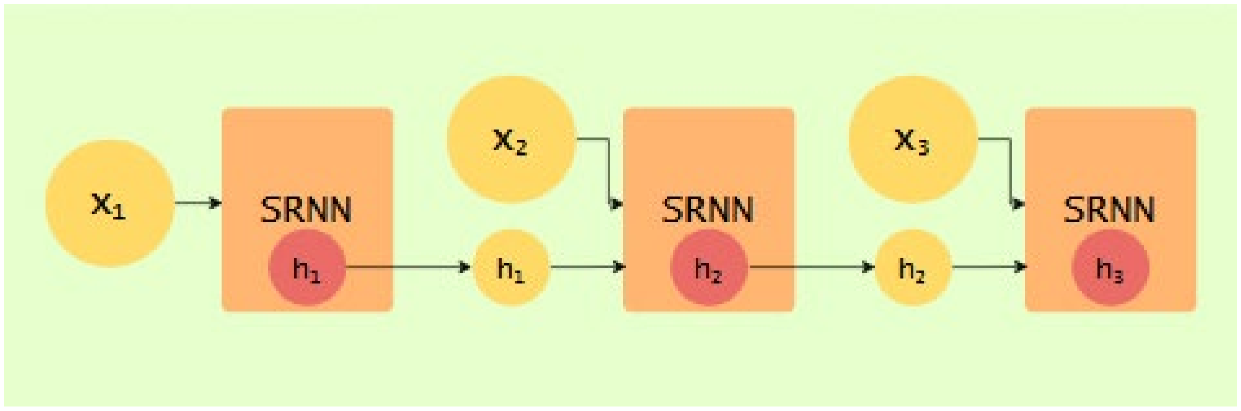


Figure 3. RNN Explicit Recurrent Representation

The idea of how the RNN is recurrent can be seen in the figure above—the simple RNN unit constructed above can be represented as multiple units connected by their hidden state. The input of size (t, k) passed into the network is decomposed into t k-sized vectors. Each vector t is passed into the network at each different unit until the very last x_t vector in the sample (in this diagram, only the first t=1, 2, 3 vectors are shown). The LSTM network and GRU network are very similar to this in the sense of a repeated layer and having a basic unit. The LSTM network is more complex than the simple recurrent neural network in that it has more parameters, and therefore more computation involved.

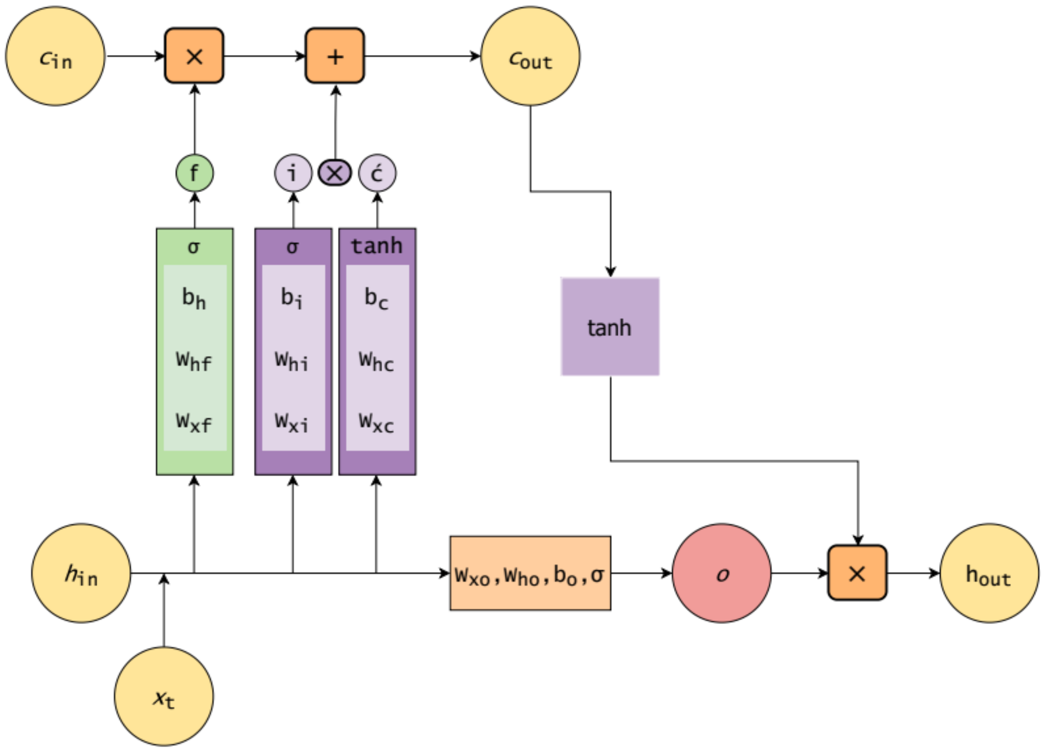


Figure 4. LSTM Structure

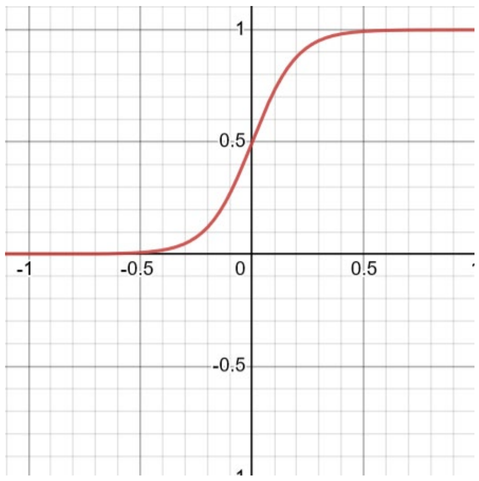


Figure 5. Sigmoid Activation

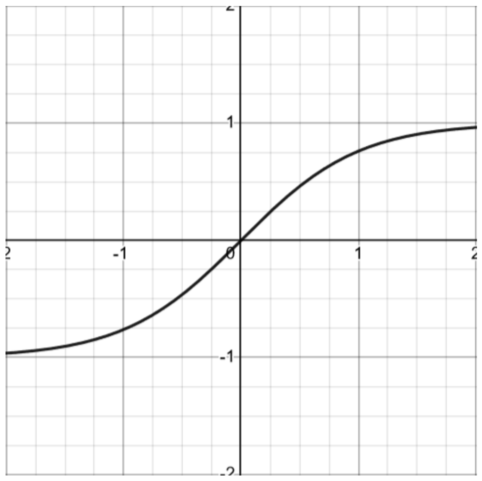


Figure 6. Hyperbolic Tangent Activation

Figure 4 depicts the internal structure of the LSTM network. The structure of the LSTM is complex, but it can be represented by a set of equations that carry out the operations illustrated in the diagram, simplifying the model into something more comprehensible. Note that x_t is a vector in the hypothetical (t, k) size sample, h_{in} is the hidden state of the last timestep, h_{out} is the produced hidden state using the current timestep, W_{ij} is a weight matrix between some matrix **i** and **j**, and the *c* is the connected state of the network. Figure 6 is the tanh function used in the third and last computation step.

1. $f = \sigma(xW_{xf} + h_{in}W_{hf} + b_f)$ (Recurrent activation)
2. $i = \sigma(xW_{xi} + h_{in}W_{hi} + b_i)$ (Recurrent activation)
3. $ć = \tanh(xW_{xc} + h_{in}W_{hc} + b_c)$ (Activation)
4. $o = \sigma(xW_{xo} + h_{in}W_{ho} + b_o)$ (Recurrent activation)
5. $c_{out} = i * ć + f * c_{in}$ (Element-wise multiplication)
6. $h_{out} = o * \tanh(c_{out})$ (Activation)

The three gates can be seen in the network: *f, i*, and *ć*. They are each responsible for controlling the cell state, c. The added gates to a recurrent neural network allow the LSTM to gain a higher level of memory functionality. The f gate, or the forget gate, learns to recognize what information is not needed in the long-term memory (the cell state) from the new data and the hidden state [4]. The input gate decides what information is worth adding to the cell state, and the ć generates a new input vector from which the *i* gate extracts information using a sigmoid activation function [5]. These three gates enable the network to learn long term features of the data through the connected state across all timesteps and short-term features through the hidden state.

3. LSTM Application to Malwares

Malwares on a typical computer have API call sequences, which are function calls that malwares make while an attack is taking place. There are hundreds of different API calls that a malware can execute. Since malware calls can be represented by function names, and there are a fixed set of calls that certain types of malwares can make, it is possible to use a certain mapping that assigns each call to a numerical value. Table 1 explains this concept below by representing each API call by a distinct ID (note that this is not reflective of the dataset associative array).

The IEEE Dataport dataset used in this research creates an associative array of the first 100 API calls that are executed by each malware: each call is represented by an integer ID ranging from 0 to 306. Each series of sequences was also either classified as goodware and malware using a 0 or 1, respectively. The data was generated through a Cuckoo Sandbox, a dynamic malware analysis software that can run malware executable files on a Linux Ubuntu host, which contains a Windows 7 machine [6]. The sandbox was used to extract the call sequences of hundreds of malware files using the call elements from the Cuckoo Sandbox reports [7]. Below in Table 2 are listed more features of the dataset directly compiled from the dataset description on IEEE Dataport [7].

Table 1. Example API Call Mapping

Malware API Call	ID
‘NtOpenThread’	000
‘ExitWindowsEx’	101
'FindResourceW'	212
'CryptExportKey'	323
'CreateRemoteThreadEx'	133
'MessageBoxTimeoutW'	134
'InternetCrackUrlW'	253

Table 2. Additional Dataset Information

Column Name	Description	Type
hash	MD5 hash of the example	32 bytes string
t_0 … t_99	API Call	Integer (0-306)
Malware	Class	Integer: 0 (Goodware) or 1 (Malware)

Note. A. Oliveira. (2019). Malware Analysis Datasets: API Call Sequences. [Table] Retrieved from https://ieee-dataport.org/open-access/malware-analysis-datasets-api-call-sequences.

From this dataset, we can identify the sequential parameters as the sets of 100 API calls that are made by each malware executable file, and therefore we can train a network based on **t = 100.** This means that each sequence that is fed into the model will be size 100. The vectors must be 2 dimensional, so the technical size is (100, 1). An LSTM network is the suitable model for the purpose of finding insights into the call sequences because it is flexible with its call size request, which can detect malware prior to its complete execution of API calls. An LSTM also allows for longer sequences of data such as in this research to be accurately evaluated due to the cell state and the gates that serve as its validators. Using this dataset required creating a balanced distribution of the goodware and malware call sequences. Prior research into this topic didn’t account for the precision and recall aspects of model results, which are just as important as the accuracy. A dataset with a large amount of call sequences classified as malware would produce misleading results if accuracy was the only metric used to validate the model. Therefore, we accounted for precision and recall by balancing the samples in both classes.

To classify the model, it is necessary to append a dense layer which uses the hidden state produced by the last timestep of the call sequence for classification. The last layer has the conventional sigmoid activation function to compress the final vector into a probability. To increase model complexity, additional hidden layers were also added in iterations of the networks. These hidden layers were part of a generic deep neural network, which has weights and biases that activate neurons in a format of several column vectors. We pursued several different network configurations, involving the LSTM and GRU networks. The single layer results for training the recurrent neural nets for 1000 epochs and optimal dense sizes are shown below in Table 3.

Table 3. Results of Single Layer RNN

Type	Hidden Layers	Precision (0, 1)	Recall (0,1)	Accuracy
GRU	1, size 8	0.84, 0.72	0.73, 0.84	0.78
LSTM	None	0.83, 0.83	0.82, 0.83	0.83
LSTM	1, size 8	0.84, 0.87	0.88, 0.82	0.85
LMTM Stacked Architecture	1, size 8	0.84, 0.91	0.91, 0.85	0.88

In Table 3, the first value of the precision and recall column is specific to a goodware classification, and the second value is specific to a malware classification. The GRU performed worse than the LSTM network in both accuracy and recall, and the last single LSTM outperformed both previous configurations due to a hidden layer which added complexity to the classification process. Although many other configurations of a single RNN layer were trained and evaluated involving a varying dense network, the firsts three networks were tuned to the optimal hidden layer size and served as a basis of the following RNN generations.

A novel approach that we used in this research to significantly improve upon the LSTM configuration used in the generations above was using the concept of a stacked-LSTM layer, which is essentially two LSTM layers of which one layer can encode the hidden state, and the other layer receives the hidden state of all **t** timesteps of the first layer and behaves like a decoder. This architecture is much more proficient at producing a hidden state sequence that the dense layer can use to deduce defining properties of malware and goodware API call sequences. This technique has been used in natural language processing (NLP), though it is a new application for malware analysis purposes. The network structure is depicted in Figure 7; in this setup, the input vector on the left-hand side is the API call sequences, for which LSTM 1 creates **t** hidden states of size 10. The hidden states are processed in LSTM 2, and the output is the last hidden state produced. By appending a typically neural network to the last LSTM layer, a probability is generated for classifying the malware.

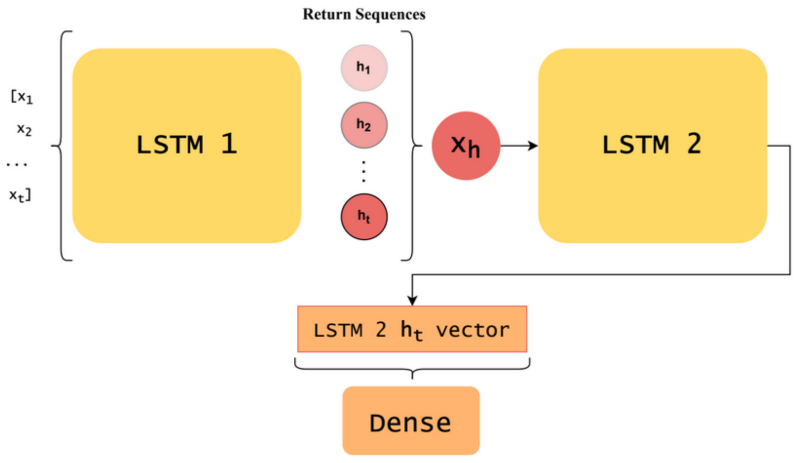


Figure 7. Structure of the Final Model

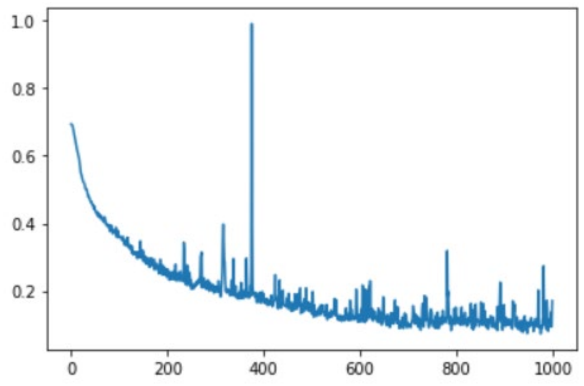


Figure 8. Stacked-LSTM Loss Graph

The stacked LSTM model with a dense size of **8** was the most successful at malware detection, having a precision of **0.84** and **0.91**, and a recall of **0.91** and **0.85**, for goodware and malware respectively. The validation accuracy was **0.88**, which is a significant finding considering that the model accuracy has the potential to improve if provided with longer API calls sequences. Figure 8 shows the loss graph for this iteration of the network while training. The trend of the loss graph is a steady decrease, though there are several epochs in which the loss spikes due to drastic gradient steps; this is normal for LSTM networks because of their customizable batch size. The results of the stacked LSTM are summarized in the last row of Table 3.

4. Integration with Modern Computers

As mentioned before, the dataset used was from a dynamical malware analysis software which runs a Windows operating system in an isolated environment. This approach to detecting a malware can be used for a local system. For the purpose of this section, we will use the Windows operating system with malicious files as generic Windows executables. Below, Figure 9 explains how the Cuckoo Sandbox service works [8].

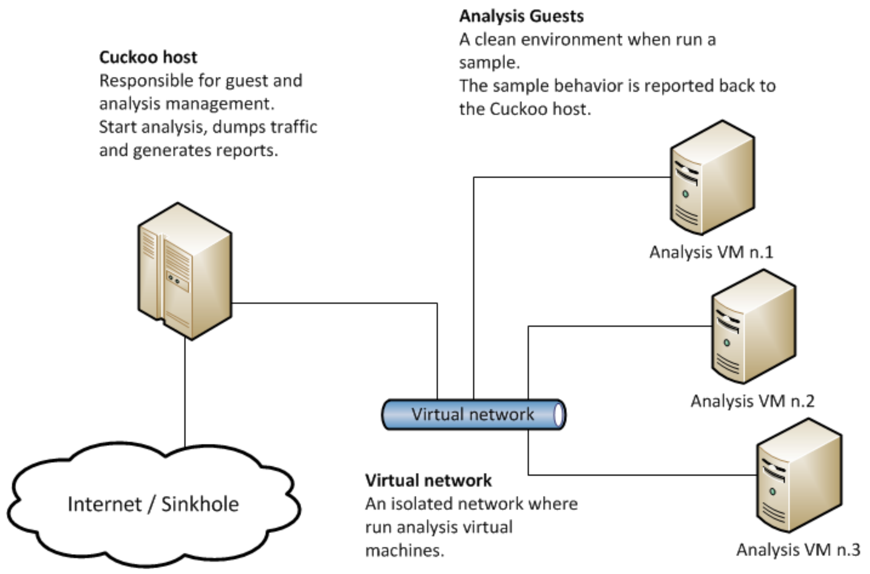

Analysis Guests
A clean environment when run a sample.
The sample behavior is reported back to the Cuckoo host.
Cuckoo host
Responsible for guest and analysis management.
Start analysis, dumps traffic and generates reports.
Analysis VM n.1
Analysis VM n.2
Analysis VM n.3
Virtual network
Internet / Sinkhole
Virtual network
An isolated network where run analysis virtual machines.

Figure 9. Cuckoo Sandbox Main Architecture

Note. Image is from the Cuckoo Sandbox Documentation [8].

The Cuckoo host is the central server which sends a request to one of the virtual machines, and the software handles extracting the API sequence calls from an executable file. The files are run on the VMs, and a report is provided back to the host computer. To apply this model to a real machine, the sandbox cannot be used because Cuckoo provides no interface for third party software. Instead, we must utilize the Windows built-in WinAPI which provides a Registry of the functions and runs the API service [9]. A tool called PEStudio can be installed to spot and extract function calls of the potential malware executables [10]. If we create an associative array, or map, as we did with the IEEE dataset, of the Windows registry, and of the functions that are invoked by the executable through PEStudio, we can predict the classification of these values based on the network's analysis. The length of the request is unrestrictive, so we may send a request for the model to process at any given time in the file's execution. Therefore, a time parameter can be provided to the network script which extracts sequences of multiple lengths over the given time before classification. If the network yields a value of 1, and therefore deems the file a malware, the process ID (PID) of the executable can be killed, effectively shutting down the thread on the CPU running the malware.

5. Summary

In this paper, we discussed recurrent neural networks and their functionality, as well as their usefulness to detect malwares using API sequence calls. We discussed how to leverage the RNN capability of finding relationships between sequences of data and providing an insightful hidden state for the basis of predictions. The final model was a stacked Long Short-Term Memory (LSTM) network with a dense layer of size 8 that accepts the hidden state of the second LSTM. The model makes a prediction using the last sigmoid layer. Lastly, we discussed how to use software tools to extract API sequence calls on a Windows computer and stop a process that is identified as a potential malware by the model.

References

[1] Glamoslija, K. (2021, March 22). *10 most dangerous virus & malware threats in 2022*. SafetyDetectives. Retrieved August 28, 2022, from https://www.safetydetectives.com/blog/most-dangerous-new-malware-and-security-threats/

[2] Hermans, M., & Schrauwen, B. (1970, January 1). *[PDF] training and analyzing deep recurrent neural networks: Semantic scholar*. undefined. Retrieved August 28, 2022, from https://www.semanticscholar.org/paper/Training-and-analyzing-deep-recurrent-neural-Hermans-Schrauwen/64cd8a192de0f4de444db759b14cadce111fd904

[3] *A guide to RNN: Understanding recurrent neural networks and LSTM Networks*. Built In. (n.d.). Retrieved August 28, 2022, from https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[4] *Co-uk*. ssla. (2020, August 20). Retrieved August 28, 2022, from https://www.ssla.co.uk/long-short-term-memory/

[5] Cheng, H.-Y., & Yu, C.-C. (n.d.). *LSTM cells. (a) forget gate, (b) input gate, (c) updating cell state* ... Retrieved August 29, 2022, from https://www.researchgate.net/figure/LSTM-Cells-a-Forget-Gate-b-Input-Gate-c-Updating-Cell-State_fig2_355833547

[6] *A guide to RNN: Understanding recurrent neural networks and LSTM Networks*. Built In. (n.d.). Retrieved August 28, 2022, from https://builtin.com/data-science/recurrent-neural-networks-and-lstm

[7] Oliveira, A. (2019, December 12). *Malware analysis datasets: API call sequences*. IEEE DataPort. Retrieved August 28, 2022, from https://ieee-dataport.org/open-access/malware-analysis-datasets-api-call-sequences

[8] *What is cuckoo?*. What is Cuckoo? - Cuckoo Sandbox v2.0.7 Book. (n.d.). Retrieved August 28, 2022, from https://cuckoo.sh/docs/introduction/what.html

[9] Windows API calls: The malware edition. (2020, April 29). Retrieved August 28, 2022, from https://sensei-infosec.netlify.app/forensics/windows/api-calls/2020/04/29/win-api-calls-1.html

[10] Winitor. (n.d.). Retrieved August 28, 2022, from https://www.winitor.com/