International Journal of Innovative
Computing, Information and Control
Volume **15**, Number **4**, August **2019**

ICIC International ©2019 ISSN 1349-4198
pp. 1413–1424

CYBERSPACE ATTACK DETECTION BASED ON ADVANCED INITIALIZED RECURRENT NEURAL NETWORK

Chaopeng Li[1,2], Yiqiang Sheng[1] and Jinlin Wang[1,2,*]

[1]National Network New Media Engineering Research Center
Institute of Acoustics, Chinese Academy of Sciences
No. 21, North 4th Ring Road, Haidian District, Beijing 100190, P. R. China
{ licp; shengyq }@dsp.ac.cn; *Corresponding author: wangjl@dsp.ac.cn

[2]School of Electronic, Electrical and Communication Engineering
University of Chinese Academy of Sciences
No. 19(A), Yuquan Road, Shijingshan District, Beijing 100049, P. R. China

Received September 2018; revised January 2019

Abstract. *The recurrent neural network (RNN) can be used to detect cyber attacks. However, during training, vanishing and exploding gradients are encountered. Thus, in this study, an RNN model with an advanced weight initialization is proposed to alleviate possible gradient problems. We considered attack detection as a classification task and adopted hierarchical RNN and multilayer perceptron (MLP) model to identify attacks. Thereafter, the causes of vanishing or exploding gradients are analyzed. Based on the distribution of cyberspace data, derivations are conducted and an improved weight initialization approach facing RNN was employed. There are two formats for RNN outputs: last step only and all steps available. Accordingly, initializations for these different formats are fine-tuned. Based on a public dataset, various learning convergences of the state-of-the-art initialization schemes, suggested for use for the past three years, are compared, and the influence of different embedding methods is discussed as well. Finally, experiments are conducted. Results show that our proposed initialization method has a lower error rate (about 9% relative decrease) than other initializations have.*
Keywords: Cyberspace attack detection, Recurrent neural network, Weight initialization

1. **Introduction.** Present day cyber attacks are increasingly becoming diversified and complex, posing serious threats to network infrastructures. The recurrent network is widely used for cyber attack detection because it has excellent effects on nonlinearity and temporality, such as the simple-RNN [1] and long short-term memory (LSTM) [2]. Buczak and Guven [3] used the RNN model for attack detection. Kim et al. [4] used LSTM for attack detection and Wang et al. [5] employed the LSTM and convolution neural network (CNN) model for intrusion detection based on network flows. All of the above studies did not consider limited computing resources and tended to use complex models such as LSTM.

In many scenarios, the cyber attack detection is employed in edge network and devices, such as intelligent switches, routers and gateways [6,7]. There are also some studies about adaptive attack detection in cloud computing [8]. The throughput of network data is considerably huge to the extent that a network device often accommodates 100-Gbps traffic or larger. Most network devices, whose computing resources are scarce, use dedicated network chips. Therefore, we anticipate the use of the lighter simple-RNN model for cyber-attack detection.

DOI: 10.24507/ijicic.15.04.1413

The simple-RNN model was proposed and considered able to express any nonlinear characteristic but it got an inferior training performance which is a result of vanishing and exploding gradients. The cause of the gradient problem in the simple-RNN model is that at each time step, the hidden layer receives a new input that causes gradients to increase. There are three kinds of methods to alleviate the gradient problem: modifying the model structures, improving training methods and weight initialization methods. For modifying the structures, such as LSTM and gate recurrent unit [9] began to be proposed, threshold structures were used, RNN models were improved, and convergence was optimized. New training methods, such as the Hessian-free (HF) [10] method, have also yielded advantageous contributions. However, the foregoing methods, whether for improving the model structure or changing the training method, all increase the complexity and amount of calculation involved. Evidently, as the LSTM model replaces the neural function with a threshold structure, more complex calculations are involved. In a follow up study on the recurrent neural network [11], it is discovered that HF is significantly unstable for the model and requires extra damping on the hidden state, making it further complicated. The third suitable method to alleviate the gradient problem is a weight initialization scheme which can sufficiently make the model converge faster without more computing resources. Previously, there have been studies on the weight initialization. For example, Glorot and Bengio [12] proposed an initialization method for MLP using Tanh and sigmoid as activation functions to avoid the gradient problem of when the model extremely deep. He et al. [13] also proposed an initialization method based on the CNN model with ReLU and obtained good optimization results. All of the above methods do not work in the simple-RNN model. Based on the simple-RNN network, Graves used a normal distribution initialization scheme with a standard deviation of 0.1 [14], but the possibility of encountering vanishing and exploding gradients remains when such a scheme is used.

The motivation of our research is to alleviate the gradient problems and to design an advanced initialization for the simple-RNN model, so we propose a new weight initialization method for the simple-RNN to optimize training models for cyber attacks. Two active functions (Tanh and ReLU) and two types of output layers (the last step only and all steps available) are involved and discussed. To confirm the initialization and the RNN-MLP model, some state-of-the-art initialization methods are involved and compared with. The experiments are based on DARPA 1998 public datasets. According to the convergence curves, the gradient problems using the proposed initialization are alleviated compared with using other methods. The experimental results show that the proposed initialization gets a 4.05% absolute decreased (Equivalent to about 9% relative reduction) error rate compared with other initialization.

The paper is organized as follows. In the second section relevant research on the RNN for cyber-attack detection and work pertaining to weight initialization are introduced. In the third section, a new scheme for weight initialization based on the simple-RNN network is proposed and the advanced weight initialization is illustrated. In the fourth section, experimental setup and results, and various influencing factors are discussed. Conclusions and future work are described in the fifth section.

2. **Detecting Attacks Based on Recurrent Neural Network.**

As for network data, especially for microflows, it is suggested that CNN or MLP be used for capturing packet features, and thereafter employ the recurrent model (e.g., simple-RNN, LSTM, and GNU) to capture flow features. Finally, a classifier is employed for discrimination and detection. These similar structures have also been mentioned in [5]. In general, the process of

generating packet features is called packet representation; the process of generating flow features is called flow representation.

An RNN-MLP model structure is employed here as a demonstration for attack detection. The weight initialization is also based on this model. Figure 1 shows the structure of RNN-MLP. The first layer from the bottom is the packet representation, while the second layer is RNN, which is the flow representation. The third layer is a softmax classifier.

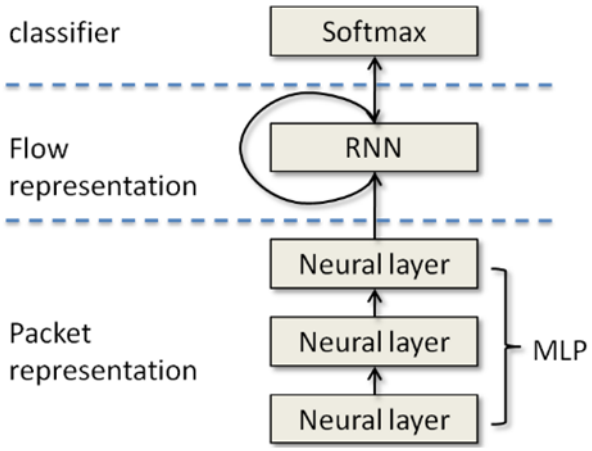

FIGURE 1. Simple-RNN-MLP model structure

2.1. **Packet representation.** The packet representation refers to the whole MLP because raw data are recommended as inputs in the deep neural network commonly. The packets of byte-level data are directly considered as model inputs. The output of MLP is presented as follows:

$$o_{mlp} = \theta\left(W_{m\_o} \cdot \theta(W_{m\_h} \cdot x + b_{m\_h}) + b_{m\_o}\right), \tag{1}$$

where x refers to input data; W_{m_o} and W_{m_h} are the weights of the output and hidden layers, respectively; b_{m_h} and b_{m_o} are the biases of the hidden and output layers, respectively; the function $\theta(\cdot)$ is the activation function; o_{mlp} is the packet feature vector.

2.2. **Flow representation.** The flow representation refers to the entire recurrent neural network [16]. There are two aspects of the inputs to a recurrent model. One part is the output of MLP, o_{mlp}, and the other is the output of the recurrent layer from the last time step. The recurrent network is presented as follows:

$$o_{rnn,t} = \theta\left(W_{r\_i} \cdot o_{mlp} + W_{r\_h} \cdot o_{r,t-1} + b\right), \tag{2}$$

where W_{r_i} and W_{r_h} are the weights of the input layer and the recurrent layer, and the symbol b is the bias; $o_{r,t}$ is the output of the recurrent layer at the t^{th} step.

3. Weight Initialization.

3.1. **A review of weight initialization.** The RNN model currently employs a common distribution initialization scheme with a variance of 0.1. Direct schemes could also be used using the initialization methods of either Glorot and Bengio or He et al. However, the above methods can still cause the gradient problem. We will analyze the main reasons for this and propose a new initialization method, as follows.

Any two-layer neural connection of a multilayer neural network can be expressed simply as

$$x_{l+1} = \theta(y_l) \text{ and } y_l = w_l x_l, \tag{3}$$

where x_l is the output of layer l, w_l is the weight of l, and y_l is the input of layer $l+1$. Here, the bias is ignored because its initialization is always zero, which cannot change the mean and variance of y_l. Then, in order to avoid the gradient problem, it is necessary to let $Var[x_{l+1}] = Var[x_l]$. For the above goal, the gradient problem can be avoided only if the initialization range of w_l is set properly.

However, for the hidden layer (recurrent layer) of RNN, the input has two parts: the input at each time step and output of the recurrent layer from the previous step. Therefore, the hidden layer is completely different from DNN (deep neural network) model in the case of forward propagation. Similarly, the output of the RNN layer has two parts. One is connected to the classifier, and the other is passed to the recurrent layer of the previous step as input. According to the chain rule if the BGD (batch gradient descent) method is used for optimization, the gradient will consist of these two parts. We will analyze the influence of the weight initialization in the forward case and backward case.

3.2. **Weight initialization derivation based on Tanh function.** Our derivation mainly follows those of [12,13]. The central concept is to balance the variance of the response at each time step. The Tanh function is an activation function proposed in 2009, expressed as follows:

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4}$$

3.2.1. *Forward propagation case based on Tanh.* The extended form of the recurrent layer in the forward case can be seen in Figure 2.

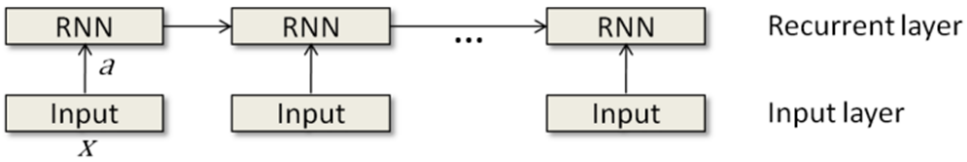

FIGURE 2. The extended format of the forward propagation

The following definitions apply: z^t is the output of one unit from the recurrent layer at the t step, and a^t means an input of one neural unit. The recurrent model can be expressed as follows:

$$a^t = \sum_{i=1}^{n_I} w_{ih} \cdot x^t + \sum_{h'}^{n_H} w_{h'h} \cdot z^{t-1}, \ \forall t, \qquad z^t = \theta(a^t). \tag{5}$$

Generally, because the neuron function is active, it can be regarded as a linear function, such as $\theta(x) = x$ and $z^t = a^t$.

In the case of the forward propagation, a^t, z^t, x^t, and w are random variables with a mean of zero. The sizes of input layer and hidden layer are respectively n_I and n_H. Accordingly, we obtain the following:

$$Var[a^t] = n_I\, Var[w_{ih}]\, Var[x^t] + n_H\, Var[w_{h'h}]\, Var\left[z^{t-1}\right], \tag{6}$$

where Var is the variance. Set the assumption of $Var[x^t] = Var[z^{t-1}] = Var[z^t] = Var[a^t]$. By considering the weights as a unified labeled with $Var[w_{ih}] = Var[w_{h'h}] \triangleq Var[w]$, the following is obtained:

$$Var[a^t] = (n_I + n_H)\, Var[w]\, Var\left[a^{t-1}\right]. \tag{7}$$

Additionally, when $t = 1$, set $w_{h'h} = 0$. Then, we have, $Var[a^1] = n_I\, Var[w_{ih}]\, Var[x^1]$. According to the purpose of the variance, with $Var[a^t] = Var[a^{t-1}]$ previously, we have

$$\begin{cases} (n_I + n_H)\, Var[w_{ih}] = (n_I + n_H)\, Var[w_{h'h}] = 1, \\ n_I\, Var[w_{ih}] = 1. \end{cases} \tag{8}$$

3.2.2. *Backward propagation case based on Tanh.* The gradient descent method is generally used to train models. For the RNN model, there are two output methods and two corresponding back propagation modes. One is to use only the last step as output, and the other is to employ the average of each step as output. The gradient derivation of the two methods is different. The RNN model structure is shown in Figure 3.

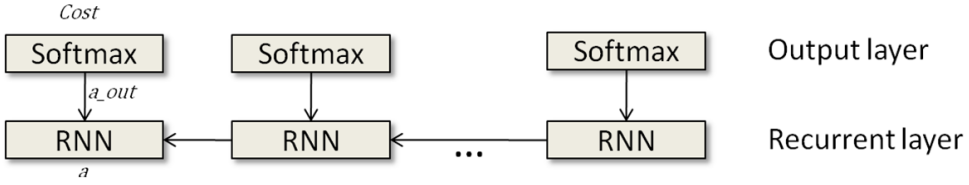

FIGURE 3. Extended format of the backward propagation

To derive the objective function, we get

$$\frac{\partial Cost}{\partial a^t} = \theta'(a^t)\frac{\partial Cost}{\partial z^t} = \theta'(a^t)\left(\sum_{k=1}^{n_K} w_{hk} \cdot \frac{\partial Cost}{\partial a_{out}^t} + \sum_{h'=1}^{n_H} w_{h'h} \cdot \frac{\partial Cost}{\partial a^{t+1}}\right), \tag{9}$$

where n_K is the size of output layer.

To avoid the gradient from disappearing or exploding, we have the following: $Var\left[\frac{\partial Cost}{\partial a^t}\right] = Var\left[\frac{\partial Cost}{\partial a^{t+1}}\right] = Var\left[\frac{\partial Cost}{\partial a_{out}^t}\right]$ and $\theta'(a^t) = 1$. From the condition $Var[w_{hk}] = Var[w_{h'h}] \triangleq Var[w]$ and then we obtain

$$Var\left[\frac{\partial Cost}{\partial a^t}\right] = (n_K + n_H)\, Var[w]\, Var\left[\frac{\partial Cost}{\partial a^{t+1}}\right]. \tag{10}$$

For the output of the RNN model, there are two cases: one is the last step only, and the other includes all available steps. For the first case, if $t = T$, then, $w_{h'h} = 0$ is obtained. If $t = 1, 2, \ldots, T-1$, then we obtain $w_{hk} = 0$. Primarily

$$\begin{cases} n_H\, Var[w_{h'h}] = 1, \\ n_K\, Var[w_{hk}] = 1. \end{cases} \tag{11}$$

Combining Equations (8) and (11), we obtain

$$\begin{cases} Var[w_{h'h}] = \dfrac{3}{(2n_H + 2n_I)}, \\ Var[w_{ih}] = \dfrac{2}{(n_H + 2n_I)}, \\ Var[w_{hk}] = \dfrac{1}{n_K}. \end{cases} \tag{12}$$

For the second case, if $t = T$, then, $w_{h'h} = 0$. If $t = 1, 2, \ldots, T-1$, then $w_{hk} \neq 0$. Primarily, we have

$$\begin{cases} (n_K + n_H)\, Var[w_{h'h}] = 1, \\ (n_K + n_H)\, Var[w_{hk}] = 1, \\ n_K\, Var[w_{hk}] = 1. \end{cases} \tag{13}$$

Combining Equations (8) and (13), we obtain the following:

$$\begin{cases} Var[w_{h'h}] = \dfrac{2}{(n_K + n_H + n_I)}, \\ Var[w_{ih}] = \dfrac{2}{(n_H + 2n_I)}, \\ Var[w_{hk}] = \dfrac{2}{(2n_K + n_H)}. \end{cases} \tag{14}$$

3.3. **Weight initialization derivation based on ReLU function.** The expression for ReLU is

$$relu(x) = \begin{cases} x, & \text{if } x > 0, \\ ax, & \text{if } x \leq 0. \end{cases} \tag{15}$$

Here, x is the input of the nonlinear activation function $relu(\cdot)$, and a is a coefficient controlling the slope of the negative part. When $a = 0$, it becomes ReLU; when a is a learnable parameter, Equation (15) is referred to as parametric ReLU. However, only ReLU is discussed in this paper.

Among the most popular activation functions at present it is ReLU. There are two areas involved: one is the area in $(0, +\infty)$, as the positive area, and the other is in $(-\infty, 0)$, as the negative area. Because some have thought that the extremely large output value of ReLU might make the gradient problem more critical, the use of ReLU has not been suggested. However, in this study, ReLU was considered as the activation method and proposed initialization scheme to mitigate the gradient problem.

3.3.1. *Forward propagation case based on ReLU.* Because the mean of ReLU is not zero, the condition $Var[w_{h'h} \cdot z^t] = Var[w_{h'h}] \cdot Var[z^t]$ cannot be obtained. Instead, according to [13], the condition is $Var[w_{h'h} \cdot z^t] = Var[w_{h'h}]E[(z^t)^2]$. If we let $w_{h'h}$ have a symmetric distribution around zero and $b_h = 0$, then, a^t has a zero mean and symmetric distribution around zero. Then we have as follows:

$$E\left[\left(z^t\right)^2\right] = E\left[\max\left(0, a^{t-1}\right)^2\right] = \frac{1}{2}\left\{E\left[\left(a^{t-1}\right)^2\right] - 0\right\} = \frac{1}{2} Var\left[a^{t-1}\right]. \tag{16}$$

Moreover, the derivation of the variance of a^t is as follows:

$$\begin{aligned} Var\left[a^t\right] &= n_I\, Var\left[w_{ih}\right]\, Var\left[x^t\right] + n_H\, Var\left[w_{h'h}\right] E\left[\left(z^{t-1}\right)^2\right] \\ &= n_I\, Var[w_{ih}]\, Var[x^t] + \frac{1}{2} n_H\, Var[w_{h'h}]\, Var\left[a^{t-1}\right]. \end{aligned} \tag{17}$$

The following derivation is similar to that using Tanh, and we can obtain as follows:

$$\begin{cases} \left(n_I + \dfrac{1}{2} n_H\right) Var[w_{h'h}] = \left(n_I + \dfrac{1}{2} n_H\right) Var[w_{ih}] = 1, \\ n_I\, Var[w_{ih}] = 1. \end{cases} \tag{18}$$

3.3.2. *Backward propagation case based on ReLU.* The derivative of ReLU is zero or one, with equal probabilities. If we let $w_{h'h}$ and w_{hk} have a symmetric distribution around zero, and combine it to Equation (9), then we obtain

$$E\left[\frac{\partial Cost}{\partial a^t}\right] = \frac{1}{2} E\left[\frac{\partial Cost}{\partial z^t}\right] = 0. \tag{19}$$

Moreover, we have

$$E\left[\left(\frac{\partial Cost}{\partial a^t}\right)^2\right] = Var\left[\frac{\partial Cost}{\partial a^t}\right] = \frac{1}{2} Var\left[\frac{\partial Cost}{\partial z^t}\right]. \tag{20}$$

Combining Equations (9) and (20), we obtain

$$Var\left[\frac{\partial Cost}{\partial a^t}\right] = \frac{1}{2}(n_K + n_H)\, Var[w]\, Var\left[\frac{\partial Cost}{\partial a^{t+1}}\right]. \tag{21}$$

Because there are two cases for RNN model outputs, introduced in the previous section, for the first case, if $t = T$, then, $w_{h'h} = 0$; if $t = 1, 2, \ldots, T-1$, then, $w_{hk} = 0$. Primarily, we have

$$\begin{cases} \frac{1}{2} n_H\, Var[w_{h'h}] = 1, \\ \frac{1}{2} n_K\, Var[w_{hk}] = 1. \end{cases} \tag{22}$$

Combining Equations (18) and (22), we obtain

$$\begin{cases} Var[w_{h'h}] = \dfrac{3}{(n_H + 2n_I)}, \\ Var[w_{ih}] = \dfrac{4}{(n_H + 4n_I)}, \\ Var[w_{hk}] = \dfrac{2}{n_K}. \end{cases} \tag{23}$$

For the second case, if $t = T$, then, $w_{h'h} = 0$; if $t = 1, 2, \ldots, T-1$, then, $w_{hk} \neq 0$. Primarily, we have

$$\begin{cases} \frac{1}{2}(n_K + n_H)\, Var[w_{h'h}] = \frac{1}{2}(n_K + n_H)\, Var[w_{hk}] = 1, \\ \frac{1}{2} n_K\, Var[w_{hk}] = 1. \end{cases} \tag{24}$$

Combining Equations (18) and (24), we obtain

$$\begin{cases} Var[w_{h'h}] = \dfrac{4}{(n_K + n_H + 2n_I)}, \\ Var[w_{ih}] = \dfrac{4}{(n_H + 4n_I)}, \\ Var[w_{hk}] = \dfrac{4}{(2n_K + n_H)}. \end{cases} \tag{25}$$

3.4. **Summary of weight initialization.** Based on the above derivations, the variance of weights is obtained by using Tanh and ReLU activation functions. Assuming that a uniform distribution for initialization is used, then, we have the distribution $w \sim U\left[-\sqrt{3\,Var[w]}, \sqrt{3\,Var[w]}\right]$. The specific initialization method is summarized in Table 1. If a Gaussian distribution is used, then, it can be calculated by simply using the mean value and variance, which is not described here.

3.5. **Network data characteristics and preprocessing methods.** The above derivations are based on two assumptions: one, all inputs are random variables with zero expectations, and two, the activation is a linear function with the condition $Var[x^t] = Var[z^t] = Var[a^t]$. The first assumption is achieved by normalization. However, it is necessary to control the variances of inputs under the second assumption. If the input range is extremely large, it may cause vanishing and exploding gradients.

As mentioned earlier, this study mainly considers the initialization scheme for network data. Road network data is composed of bytes; the range of bytes is an integer value from 0 to 255, which is similar to pixels of images. However, network data is mainly generated during communication. In this study, an interesting phenomenon was discovered – the

TABLE 1. Uniform distribution of Tanh and ReLU

Neural function	Recurrent output at last step	Recurrent outputs at all steps
Tanh	$\begin{cases} w_{h'h} \sim U\left[-\sqrt{\frac{9}{(2n_H+2n_I)}}, \sqrt{\frac{9}{(2n_H+2n_I)}}\right] \\ w_{ih} \sim U\left[-\sqrt{\frac{6}{(n_H+2n_I)}}, \sqrt{\frac{6}{(n_H+2n_I)}}\right] \\ w_{hk} \sim U\left[-\sqrt{\frac{3}{n_K}}, \sqrt{\frac{3}{n_K}}\right] \end{cases}$	$\begin{cases} w_{h'h} \sim U\left[-\sqrt{\frac{6}{(n_K+n_H+n_I)}}, \sqrt{\frac{6}{(n_K+n_H+n_I)}}\right] \\ w_{ih} \sim U\left[-\sqrt{\frac{6}{(n_H+2n_I)}}, \sqrt{\frac{6}{(n_H+2n_I)}}\right] \\ w_{hk} \sim U\left[-\sqrt{\frac{6}{(2n_K+n_H)}}, \sqrt{\frac{6}{(2n_K+n_H)}}\right] \end{cases}$
ReLU	$\begin{cases} w_{h'h} \sim U\left[-\sqrt{\frac{9}{(n_H+2n_I)}}, \sqrt{\frac{9}{(n_H+2n_I)}}\right] \\ w_{ih} \sim U\left[-\sqrt{\frac{12}{(n_H+4n_I)}}, \sqrt{\frac{12}{(n_H+4n_I)}}\right] \\ w_{hk} \sim U\left[-\sqrt{\frac{6}{(2n_K+n_H)}}, \sqrt{\frac{6}{(2n_K+n_H)}}\right] \end{cases}$	$\begin{cases} w_{h'h} \sim U\left[-\sqrt{\frac{12}{(n_K+n_H+2n_I)}}, \sqrt{\frac{12}{(n_K+n_H+2n_I)}}\right] \\ w_{ih} \sim U\left[-\sqrt{\frac{12}{(n_H+4n_I)}}, \sqrt{\frac{12}{(n_H+4n_I)}}\right] \\ w_{hk} \sim U\left[-\sqrt{\frac{12}{(2n_K+n_H)}}, \sqrt{\frac{12}{(2n_K+n_H)}}\right] \end{cases}$

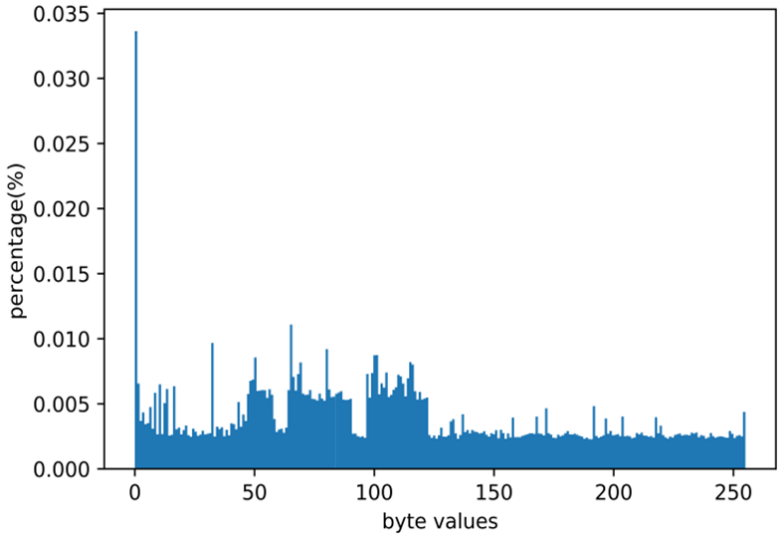

FIGURE 4. The distribution of network data

distribution of each value of the network data is not uniform. The specific distribution is shown in Figure 4. Based on statistical results, we found that integers 0 to 9, the ASCII values of the letters A to z, appear more frequently than other values, indicating that network data are mainly generated by communication among people. Furthermore, the frequency of 0 is considerably high. This is because of the extensive use of reserved fields and zero-value padding in some protocols. The network data distribution can be abstractly expressed as follows:

$$P(x) = \begin{cases} a, & x = 0 \\ b, & x = [48, 49, \ldots, 57] \cup [65, 66, \ldots, 90] \cup [97, 98, \ldots, 122] \\ c, & x = \text{else} \end{cases}, \tag{26}$$

where a is the probability of the occurrence of 0, b is the probability of the occurrence of ACSII code values corresponding to the letters A to z, and c is the probability of other values.

According to certain real network data (e.g., DARPA 1998 dataset), we used the values $a \approx 0.033$, $b \approx 0.005$, and $c \approx 0.003$, and the average is 109.3. Based on these coefficients, the normalization can be designed as $\hat{x} = 2 \cdot \frac{x - x_{mean}}{x_{\max}}$.

Additionally, embedding can be employed to recode the inputs when a deep network is used for training, such as one-hot embedding and distribution embedding. In network data, bytes are regarded as tags rather than values, similar to word representations. Hence, when processing, network data employing one-hot or distributed embedding is suggested. The specific details of experimental results are discussed in the following section.

4. Experiments.

In this study, sufficient experiments were conducted to evaluate our proposed method and compare it with other detection approaches: (1) Firstly, datasets and metrics are illustrated; (2) Secondly, different initialization methods are compared; (3) Finally, different embedding schemes are analyzed.

4.1. Datasets and metrics.

As previously mentioned, only the application of a simple-RNN structure was considered based on cyberspace data. A network microflow is a sequence of packets identified by five tuples. Because of the ability of the recurrent model to learn temporal information between packets, it is suitably employed for microflows.

As test subjects, public datasets of DARPA 1998 were used. These datasets were sponsored by DARPA for the first realistic and systematic evaluation of a research intrusion detection system, published by MIT Lincoln Laboratory in the United States in 1998 [15]. Datasets contain a seven-week train set and two-week test set. In the dataset, traffic data contain four types of attacks, including DoS, Probe, U2R, and R2L. The percentage of attacks in the train set of DARPA 1998 is approximately 65.54%, whereas the proportions of attacks in DARPA are 63.29%, 1.99%, 0.26%, and 0.01%. The proportion of the test set is similar to that of the train set.

In order to verify the effectiveness of the initialization scheme, error rates are used. When training the model, its convergence can be adjudged based on the error rate. The experiments described as follows are all based on error rates.

4.2. Comparison of different initialization schemes.

Currently, mainstream initialization methods were proposed by He et al., Glorot and Bengio, Hinton et al., and Graves. These initialization methods were compared with our method in this study.

1) Method by He et al.

The initialization proposed by He et al. [13] is based on the ReLU activation function. The method was verified and initialized according to the CNN model and Gaussian distribution, respectively, using a variance of $\frac{1}{2} n\, Var[w] = 1$.

2) Method by Glorot and Bengio

The initialization proposed by Glorot and Bengio [12] is based on the Tanh activation function. The method was verified and initialized according to the common DNN model and uniform distribution with a variance of $(n_{in} + n_{out})\, Var[w] = 2$, respectively.

3) Method by Hinton et al.

Hinton at el. [11] presented the initialization using an identity matrix scaled by 0.01 for the recurrent matrix.

4) Method by Graves

Graves [14] made the initialization based on a flat random distribution in the range $(-0.1, 0.1)$, or a Gaussian distribution with a mean of 0 and standard deviation of 0.1.

Because the initialization method proposed by Glorot and Bengio is based on the Tanh activation function, and that proposed by He et al. is based on the ReLU function, we

designed two sets of experiments. The methods of Hinton et al. and Graves do not limit the type of activation.

In the experiments, the current layer contains 512 units. The structure of the whole model is listed in Table 2. The training is terminated when the curve has converged or there is an excess of 500,000 mini-batch iterations. The experimental results are shown in Figure 5.

It can be observed in Figure 5(a) that the curve of the proposed method performed better than others did. Even curves using the initializations of Graves and Hinton et al. did not converge as well. The curve of Glorot and Bengio did converge but encountered a vanishing gradient after 80 iterations. The experimental setup of the all-step output format using the Tanh function is shown in Figure 5(b). Our curve converged faster than did all the others. In Figure 5(c), our curve and that of He et al. converged. However, the former curve is smoother than the latter is. Moreover, the other two curves could not converge. In Figure 5(d), although our curve converged, that of He et al. converged

TABLE 2. Comparison with other methods by error rate (%)

	Tanh + all	Tanh + last	ReLU + all	ReLU + last
He et al.	–	–	7.71	6.29
Glorot and Bengio	12.19	20.62	–	–
Hinton et al.	50.36	88.65	88.65	88.65
Graves	55.46	88.65	88.65	84.90
Ours	8.89	16.57	5.66	5.72

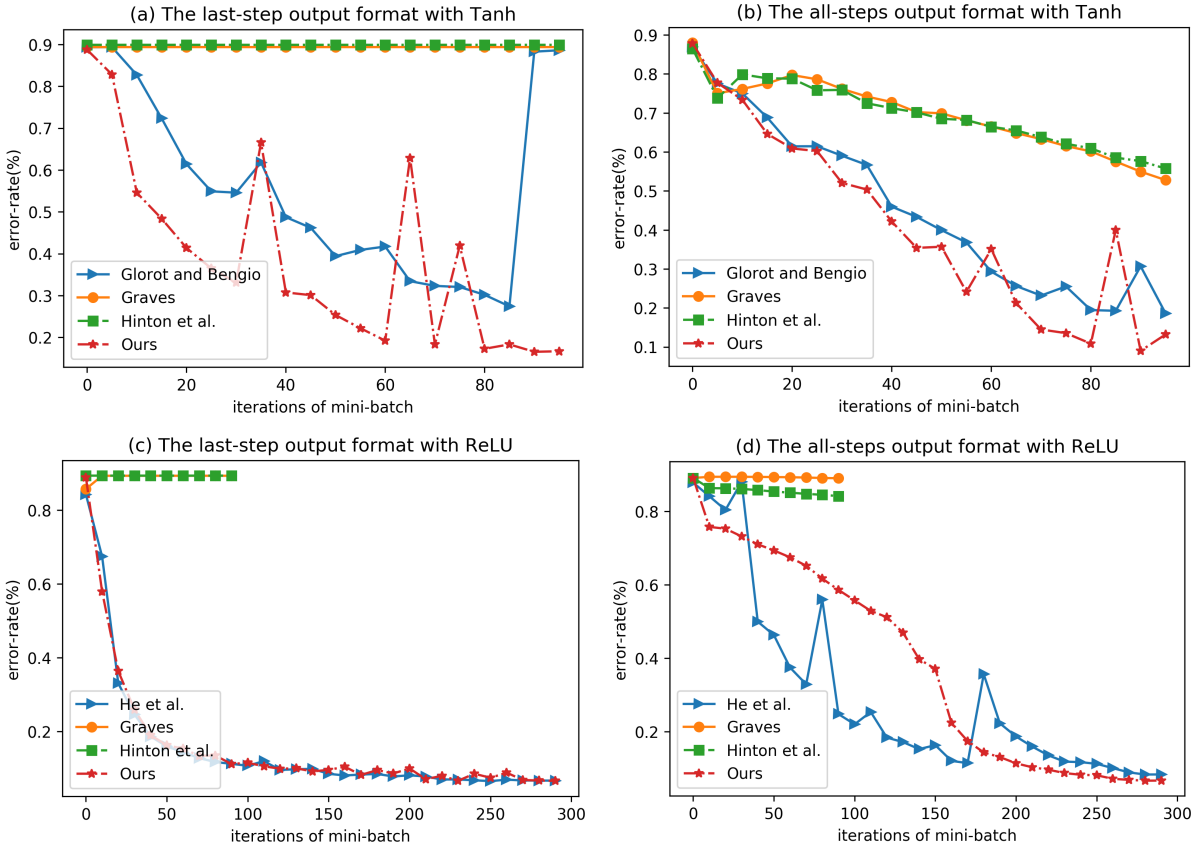

FIGURE 5. Convergence with Tanh and ReLU in different output formats

faster. Nevertheless, at approximately 170 iterations, vanishing gradients occurred and continued converging up to 300 iterations. The other two methods did not converge.

The final best two-category accuracies of each method are presented in Table 2. Note that the accuracies with Tanh are based on the test set with 100 iterations training and the accuracies with ReLU are with 300 iterations training. It is obvious that under all of the situations, our initialization did a better job than other methods. The lowest error rate is under the situation of Tanh and last step only, which exceed 4.05% absolute reduction than the second best method (Glorot and Bengio's). The absolute reductions of the error rates are 3.30%, 4.05%, 2.06% and 0.57% (the corresponding relative reductions are 27.07%, 19.64%, 26.71% and 9.06%) compared with the second best method at the different situations, respectively.

In conclusion, because vanishing or exploding gradients practically never occurred in our curve, our initialization is stable and available. However, occasionally, convergence is difficult because layer sizes are malformed (e.g., the recurrent size is significantly smaller than input and output sizes).

In previous discussion, the byte representation in embedding is introduced. However, there are practically no studies involved in the coding method for inputs. Consequently, we do not have conclusive knowledge to identify the best type of embedding. In this study, we investigated distributed embedding and one-hot embedding, and analyzed the performances of these two encoding methods.

In Figure 6, it is evident that the distributed embedding curve has a better convergence than the other two types have. The one-hot embedding curve also converges fast. Actually, the distributed embedding structure corresponds to adding a layer at the bottom of the model that is deeper than other methods are. Additionally, the one-hot embedding technique has spare inputs suitable for deep networks. In several literature, such as [5], the one-hot method is generally employed and has proved to be advanced.

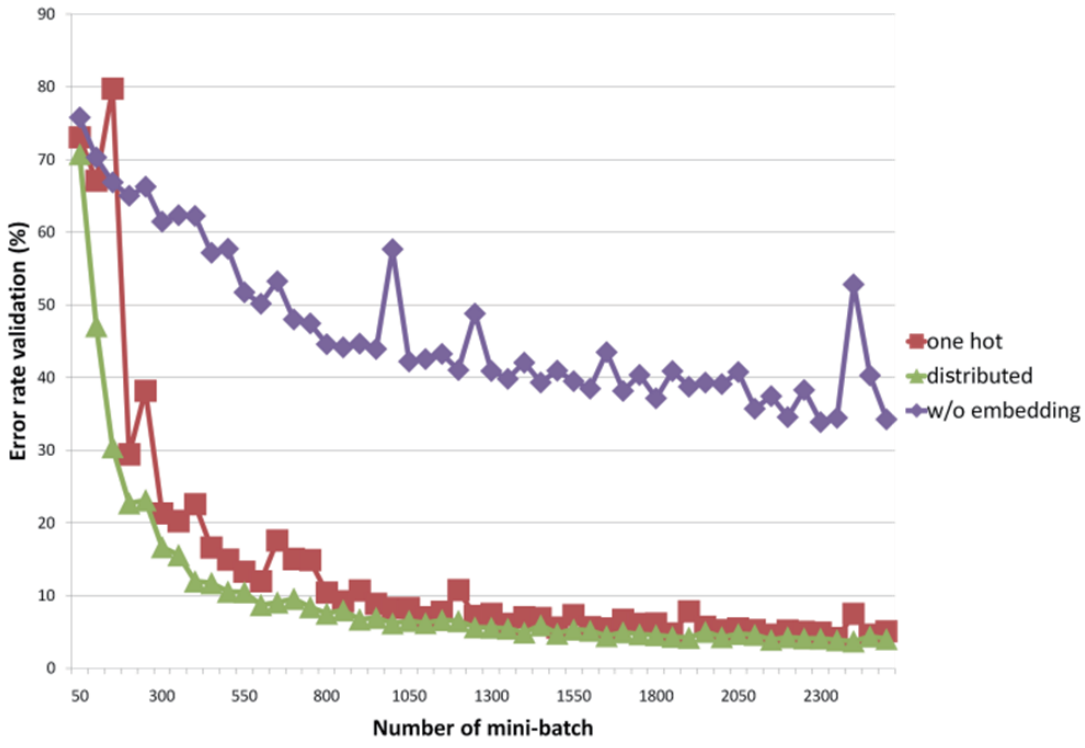

FIGURE 6. Convergence of different embedding methods

5. **Conclusion and Future Work.** The advanced initialization we proposed for the simple-RNN model is effective and the motivation, alleviating the gradient problems during training, has been achieved. We got a great converged curve compared with other state-of-the-art methods. Results have shown that our proposed initialization method has a lower error rate (about 9% relative decrease) than other initializations have. In future work, we aim to investigate other new initialization methods for simple-RNN based on new neural functions such as SeLU and ELU. Also, we aim to investigate other simple and light cyber attack detecting models facing limited computing resources.

Acknowledgment. This work was supported by National Science and Technology Major Project of the Ministry of Science and Technology, China, Grant No. 2017ZX03001019 and "First Plan" Project of Institute of Acoustics, Chinese Academy of Sciences, Grant No. SXJH201609.

REFERENCES

[1] Y. Gal and Z. Ghahramani, A theoretically grounded application of dropout in recurrent neural networks, *Advances in Neural Information Processing Systems*, pp.1019-1027, 2016.

[2] R. C. Staudemeyer and C. W. Omlin, Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data, *Proc. of the South African Institute for Computer Scientists and Information Technologists Conference*, pp.218-224, 2013.

[3] A. L. Buczak and E. Guven, A survey of data mining and machine learning methods for cyber security intrusion detection, *IEEE Communications Surveys & Tutorials*, vol.18, no.2, pp.1153-1176, 2016.

[4] J. Kim, J. Kim, H. L. T. Thu and H. Kim, Long short term memory recurrent neural network classifier for intrusion detection, *International Conference on Platform Technology and Service (PlatCon)*, pp.1-5, 2016.

[5] W. Wang, Y. Sheng, J. Wang, X. Zeng, X. Ye, Y. Huang and M. Zhu, HAST-IDS: Learning hierarchical spatial-temporal features using deep neural networks to improve intrusion detection, *IEEE Access*, vol.6, pp.1792-1806, 2018.

[6] L. Feinstein, D. Schnackenberg, R. Balupari et al., Statistical approaches to DDoS attack detection and response, *Proc. of the DARPA Information Survivability Conference and Exposition*, vol.1, 2003.

[7] M. M. Rathi, V. K. Jain, S. T. Merchant et al., *Method and System for Detecting and Preventing Access Intrusion in a Network*, U.S. Patent No. 8,707,432, 2014.

[8] G. Song, B. Li and X. Lu, Research on adaptive detection network attacks based on the improved data mining algorithm in cloud computing, *ICIC Express Letters*, vol.10, no.8, pp.1957-1964, 2016.

[9] K. Cho, B. Van Merriënboer, C. Gulcehre et al., Learning phrase representations using RNN encoder-decoder for statistical machine translation, *arXiv:1406.1078*, 2014.

[10] J. Martens, Deep learning via Hessian-free optimization, *ICML*, vol.27, pp.735-742, 2010.

[11] Q. V. Le, N. Jaitly and G. E. Hinton, A simple way to initialize recurrent networks of rectified linear units, *arXiv:1504.00941*, 2015.

[12] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, *Proc. of the 13th International Conference on Artificial Intelligence and Statistics*, pp.249-256, 2010.

[13] K. He, X. Zhang and S. Ren, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, *Proc. of the IEEE International Conference on Computer Vision*, pp.1026-1034, 2015.

[14] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, http://books.google.com/books, 2012.

[15] R. Lippman, R. Cunningham, D. Fried et al., *Results of the DARPA 1998 Offline Intrusion Detection Evaluation*, https://ll.mit.edu/ideval/files/RAID_1999a.pdf, 1998.

[16] Y. Hao, Y. Sheng, J. Wang and C. Li, Network security event prediction based on recurrent neural network, *Journal of Network New Media*, vol.5, pp.54-58, 2017.