Detecting Fraudulent Behavior Using Recurrent Neural Networks

Yoshihiro Ando[1,2,a),b)] Hidehito Gomi[2,c)] Hidehiko Tanaka[1,d)]

Abstract: Due to an increase in illegal accesses to Internet services, detecting and preventing them has become important. To prevent an illegal access, not only rule-based techniques but also machine learning techniques have been used. In the field of security, such as malware detection, machine learning techniques are used to learn behavioral patterns and detect those that are suspicious. However, there are few studies targeting the behavioral patterns of a malicious user engaged in fraudulent acts on Internet services. In this paper, we propose the approach which uses the patterns in web access logs to detect fraudulent behaviors. We apply and evaluate a recurrent neural network (RNN) to recognize these behaviors. Our result indicates that RNN is very effective for fraudulent behavior detection.

Keywords: Fraud Detection, Machine Learning, Recurrent Neural Network

1. Introduction

According to a survey conducted by the Center for Strategic & International Studies, at least $375 billion is lost globally each year as a result of cybercrime [11]. This survey includes indirect costs, such as the leakage of personal information and intellectual property theft. Counting only those losses caused by online credit card fraud, another survey concluded that at least $4.2 billion was lost in 2010 [1]. In such situations, several machine learning techniques, such as neural networks, have been applied to detect and prevent credit card fraud [13].

The previous studies of credit card fraud used transaction logs instead of the entire logs left by a malicious user on the targeted site. And these studies were not intended to model the behavior of malicious users. In adition, credit card logs are so sensitive that they are not easy to use.

Considering web logs as time series data, there are several techniques with which they can be modeled: Hidden Markov models (HMMs), state space models (SSM), and recurrent neural networks (RNN) among them. Of these, the RNN is promising in the context of deep learning. RNNs are used for modeling sequential data, such as natural language, speech, and handwritten characters. We consider that RNNs have also been effective in modeling behavioral patterns using web server logs.

This paper proposes a novel machine learning approach to detecting online credit card fraud through fraudulent behaviors. After analyzing web server log data from users who attempted illegal purchases, provided by Yahoo JAPAN, we found a characteristic pattern in the log. The behavior of the malicious user who intends to use a stolen credit card number to buy prepaid cards, such as Amazon gift cards, contains search trials in the auction service. We applied an RNN to the web log data left by a malicious user who tried to use a stolen credit card on an online auction service and evaluated its effectiveness. As a result, we obtained a model that could detect a malicious user who intended to use a stolen credit card to buy prepaid cards.

[1] Institute of Information Security
[2] Yahoo Japan Corporation
[a)] dgs165102@iisec.ac.jp
[b)] yoandou@yahoo-corp.jp
[c)] hgomi@yahoo-corp.jp
[d)] tanaka@iisec.ac.jp

2. Related Work

The RNN is a kind of artificial neural network that has several subtypes. One basic type of RNN is called a Hopfield network, which was developed by John Hopfield in 1982 [9]. A Hopfield network is not generic; it has a symmetrical structure, but cannot process sequential data.

The Elman network is another type of RNN, developed by Jeff Elman in 1990 [4]. This network type consists of 3 layers (Input, Hidden, and Output) with circular connections between units in the hidden layer. These connections take the output value of the previous time from the unit in the same hidden layer as input of the current time. This network structure reflects the reciprocal influence of sequential ordered data.

The Jordan network is another type of RNN, which was developed by Michael I. Jordan in 1986 [10]. The Jordan network is similar to the Elman network, except that its recurrent connections are between units in the input and output layers.

Elman networks have generally been shown to be superior to Jordan networks. Thus, in this paper, our approach

is based on Elman networks.

There are some applications of RNN in the field of natural language processing. Bengio et al. use an RNN to learn the context of sentence. Their proposed approach resulted in better precision than traditional the n-gram model [2]. Mikolov et al. also use RNN for learning the context of phrases and compared it to the n-gram model [12].

Speech recognition is also an important application of RNNs. Graves et al. use an RNN with long short-term memory (LSTM) to model spoken language [6]. LSTM is a special processing unit in the RNN hidden layer developed by Hochreiter and Schmidhuber in 1997 [8]. LSTM can take important parts of the sequence data and dismiss data that are not important. This feature facilitates convergence in the training phase and avoids the vanishing and exploding gradient problems.

Handwritten character recognition is another field to which RNNs have been successfully applied. Graves et al. use an RNN for on-line handwriting recognition. They train the RNN for sequence labeling to track the movement of the pen. Their proposed approach recorded better precision than commonly used HMM approaches [5]. Graves et al. also use an RNN for off-line handwriting recognition [7].

In the security literature, RNN applications are few. Sheikhan et al. use an RNN to improve IDS detection precision. They use a customized RNN, called a reduced-size RNN, for misuse-based IDS [15].

As a measure for credit card fraud detection, there are several studies that use ordinary feed-forward neural networks, including that by Patidar et al. [13]. However, literature using RNNs for this purpose does not exist to the best of our knowledge.

In the security research field, particularly in malware detection, behavior-based approaches have recently been used. Burguera et al. collect behavioral information from the Android platform using crowdsourcing and apply a k-means clustering approach to detect malware [3]. Rieck et al. use machine learning techniques of clustering and classification to learn the behavior of malware and to detect it automatically [14]. However, these researches are not intended to detect the behavior of malicious users, but of malicious software. Furthermore, these research do not model the order of actions, in contrast with our approach.

3. Our Approach

Our RNN consists of 3 layers: input, hidden, and output layers. The number of units in the input layer varies according to the features we use. The output layer has two units that each represents a possible classification. The parameters of the RNN are learned using a gradient descent method with backpropagation similar to that of feed-forward neural networks. For RNNs, there are two major learning algorithms: backpropagation through time (BPTT) [16] and real-time recurrent learning (RTRL) [17]. We use BPTT because it has a simple structure and learns fast.

It is difficult to train an RNN on a large amount of time series data. Many epochs cause difficulty in learning, as in deep learning. Therefore, several techniques, such as LSTM, which is detailed later, are used to avoid problems like vanishing and exploding gradients.

3.1 RNN

A typical RNN has three layers: input, output, and hidden layers of fully connected units. Figure 1 and 2 depicts this basic structure, which is adopted for our RNN.

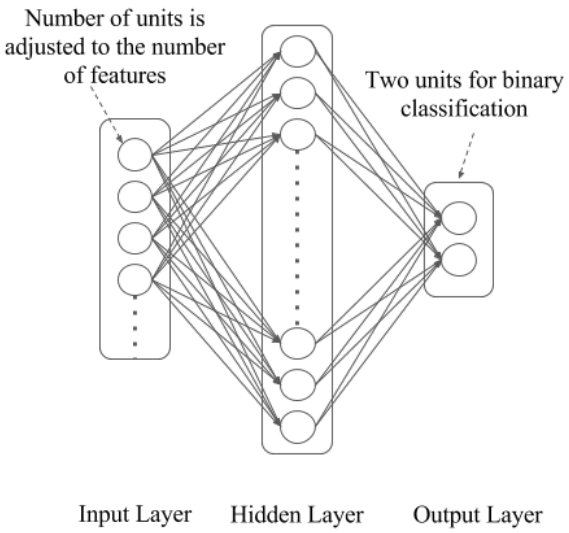

Fig. 1 RNN Structure

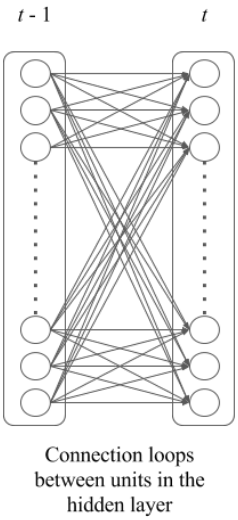

Fig. 2 RNN Hidden Layer Structure. t is the current time and $t-1$ is the previous time. These two layers are the same.

3.2 Data Encoding

Neural network uses mathematical formulas internally. Consequently, target data must be expressed as a numerical value. We use web access logs to detect fraudulent behavior, which are generally recorded as text data. Therefore,

we must encode the logs with numerical values before processing.

One approach to encoding text data for machine learning is based on the presence or absence of a given text element. Assuming that the existence of a specific word in the target text is one feature, we can represent it in binary. For example, if the text “foo” exists in the target text, we can encode the existence of the word “foo” as 1. Figure 3 shows our approach to encoding web access logs.

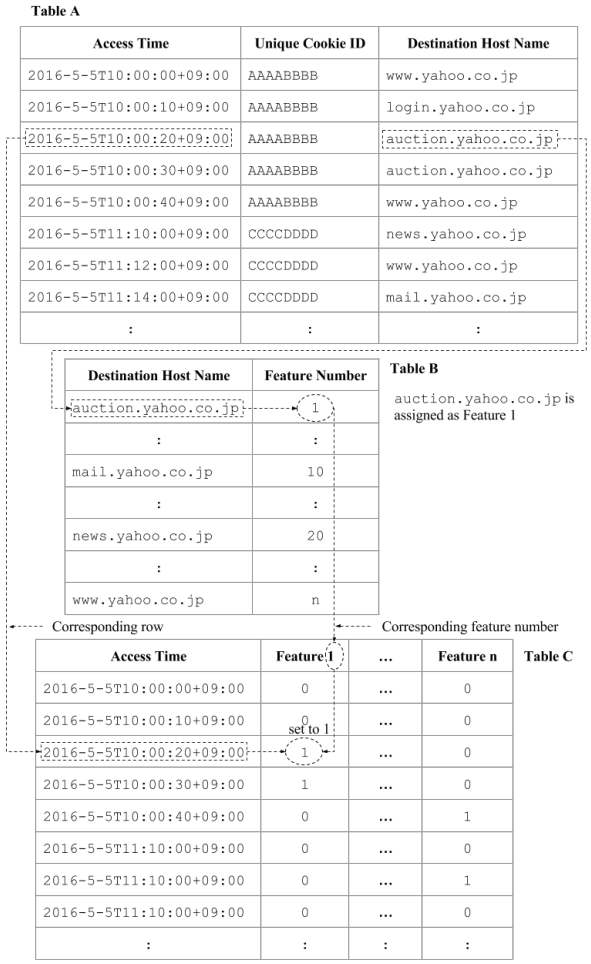

Fig. 3 Data Encoding. Table A has original logs. Access Time is the time that the user accessed. Unique Cookie ID is the value of the browser cookie issued uniquely to each browsers. Destination Host Name is the host name to which the user accessed. Table B has conversion data which converts the host name to the corresponding number of the feature field. Table C is the result of data encoding.

Table A contains a section of the web access log, featuring access date and time, a unique cookie ID, and the destination host name, from left to right. These are listed in ascending order by time. Table B shows a corresponding list of host names and feature numbers. Using Tables A and B, we can obtain the training data. For example, in Table A, the 3rd row has a value of ‘auction.yahoo.co.jp’ for the targeted host and so the corresponding column of this feature is set to 1 in Table C, and the remaining columns related to other features are set to 0.

3.3 Learning

We use BPTT to train our network. Figure 4 shows the BPTT algorithm overview.

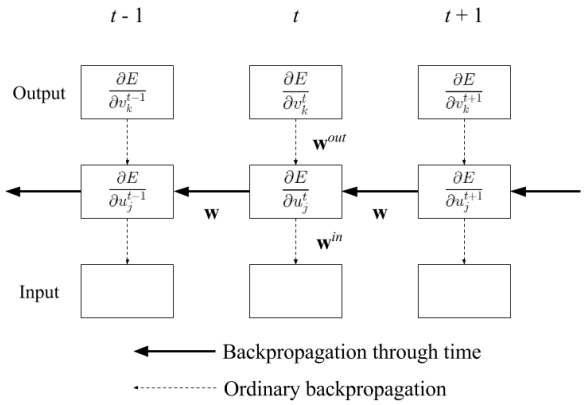

Fig. 4 BPTT. t is the current time. $\frac{\partial E}{\partial u_j^t}$ is the partial differential of the error function with the input value to the jth unit in the hidden layer. $\frac{\partial E}{\partial v_j^t}$ is the partial differential of the error function with the input value to the kth unit in the output layer. $\mathbf{w}$ is the parameter matrix.

Our goal in training an RNN is minimizing Eq. 1. n is the index for each data point in the dataset and t is the time step. k is the index of the unit in the output layer. $\mathbf{w}$ is a parameter matrix containing connection weights. d is the expected value of the specific unit corresponding to the input data. $\mathbf{x}_n$ is the input values vector of nth input.

$$E(\mathbf{w}) = -\sum_n \sum_t \sum_k d_{nk}^t \log y_k^t(\mathbf{x}_n; \mathbf{w}) \tag{1}$$

In the feed-forward phase, the input value of the units in the hidden layer is represented by Eq. 2. In Eq. 2, u_j^t is the input value of the jth unit of the hidden layer at time step t. i is the index of the unit in the input layer. w_{ji} is the weight of the connection between the ith unit of the input layer and the jth unit of the hidden layer. x_i^t is the input value of the ith unit in the input layer at time step t. j' is the index of the unit in the hidden layer, used to distinguish it from j. $w_{jj'}$ is the weight of the recurrent connection between units in the hidden layer. $z_{j'}^{t-1}$ is the output value of the j'th unit in the hidden layer at time step $t-1$.

$$u_j^t = \sum_i w_{ji} x_i^t + \sum_{j'} w_{jj'} z_{j'}^{t-1} \tag{2}$$

We use a sigmoid function as the activation function of units in the hidden layer. The sigmoid function is represented by Eq. 3:

$$f(x) = \frac{1}{1+e^{-x}} \tag{3}$$

By using a sigmoid function, we obtain Eq. 4 as the output value of the jth unit in the hidden layer.

$$z_j^t = f(u_j^t) \tag{4}$$

We can then obtain the input value to the unit in the

output layer by Eq. 5. w_{kj} is the weight of the connection between the jth unit in the hidden layer and kth unit in the output layer. z_j^t is the output value of the jth unit in the hidden layer at time step t.

$$v_k^t = \sum_j w_{kj} z_j^t \tag{5}$$

Eq. 6 conducts a classification. y_k^t is the output of the kth unit in the output layer at time step t. v_k^t is the input value to the kth unit in the output layer at time step t. L is the number of units in the output layer. We can determine which is the most likely class by the ouput value of the unit in the hidden layer.

$$y_k^t = \frac{\exp(v_k^t)}{\sum_{l=1}^{L} \exp(v_l^t)} \tag{6}$$

Backpropagation starts with calculating the partial differential of the error function with the input value of the unit in the output layer. Eq. 7 shows the value of the partial differential of the error function with the input value of the unit in the output layer.

$$\frac{\partial E}{\partial v_k^t} = y_k^t - d_k^t \tag{7}$$

v_k^t is the input value of the kth unit in the output layer at time step t. d_k^t is the target value of the kth unit in the output layer at time step t. The target values are one or zero specified in the training data. The value one means the unit should be the best unit represents that the data is fraudulent or not. For example, if the 0th unit is corresponded to the label of fraudulent and the input data is fraudulent, the 0th unit should have value one. The value zero means the unit should not be the best unit.

The partial differential of the error function with the input value of the jth unit in the hidden layer at time step t is written as:

$$\frac{\partial E}{\partial u_j^t} = \left(\sum_k w_{kj} \frac{\partial E}{\partial v_k^t} + \sum_{j'} w_{j'j} \frac{\partial E}{\partial u_{j'}^{t+1}} \right) f'(u_j^t) \tag{8}$$

The partial differential of the error function with the connection weight between the units in the input and hidden layers is represented as:

$$\begin{aligned} \frac{\partial E}{\partial w_{ji}} &= \sum_{t=1}^{T} \frac{\partial E}{\partial u_j^t} \frac{\partial u_j^t}{\partial w_{ji}} \\ &= \sum_{t=1}^{T} \frac{\partial E}{\partial u_j^t} x_i^t \end{aligned} \tag{9}$$

The partial differential of the error function with the connection weight of the internal connections in the hidden layer is represented as:

$$\begin{aligned} \frac{\partial E}{\partial w_{jj'}} &= \sum_{t=1}^{T} \frac{\partial E}{\partial u_j^t} \frac{\partial u_j^t}{\partial w_{jj'}} \\ &= \sum_{t=1}^{T} \frac{\partial E}{\partial u_j^t} u_j^{t-1} \end{aligned} \tag{10}$$

Similarly to the above, the partial differential of the error function with the connection weight between the unit in the hidden and output layers is represented as:

$$\begin{aligned} \frac{\partial E}{\partial w_{kj}} &= \sum_{t=1}^{T} \frac{\partial E}{\partial v_k^t} \frac{\partial v_k^t}{\partial w_{kj}} \\ &= \sum_{t=1}^{T} \frac{\partial E}{\partial v_k^t} z_k^t \end{aligned} \tag{11}$$

We then subtract the difference from the old weight using these differences and set as the new connection weight. Updating weight by gradient descent is represented as Eq. 12, 13, and 14.

$$w_{ji}^{new} = w_{ji} - \varepsilon \frac{\partial E}{\partial w_{ji}} \tag{12}$$

$$w_{jj'}^{new} = w_{jj'} - \varepsilon \frac{\partial E}{\partial w_{jj'}} \tag{13}$$

$$w_{kj}^{new} = w_{kj} - \varepsilon \frac{\partial E}{\partial w_{kj}} \tag{14}$$

w_{ji}^{new} and w_{ji} are the updated and current value of the connection weight between the units in the input and hidden layers. $w_{jj'}^{new}$ and $w_{jj'}$ are the updated and current value of the recurrent connection weight between the units in the hidden layer. w_{kj}^{new} and w_{kj} are the updated and current value of the connection weight between the units in the hidden and output layers. ε is called a learning coefficient.

3.3.1 LSTM

The LSTM was developed by Hochreiter and Schmidhuber in 1997 [8]. It can ease the difficulty of learning long sequences of data when using backpropagation and can also suppress the vanishing and exploding gradient problems.

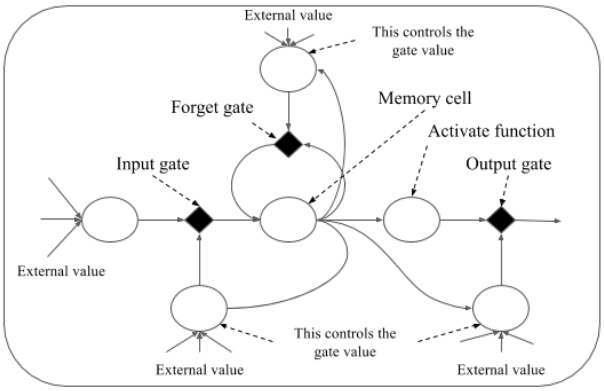

Fig. 5 LSTM. Memory cell has the value of the previous time step. Forget gate determines whether the previous value is used or not.

The internal structure of the LSTM is shown in Figure 5. We used the LSTM as a control against which we could evaluate our RNN. The result of this evaluation is discussed in the following section.

4. Evaluation

4.1 Dataset

Web access logs provided by Yahoo JAPAN were used as an evaluation dataset. These logs are those of both malicious and normal users. The logs were recorded from April 29, 2016 to May 5, 2016. Malicious users had stolen credit cards and tried to buy prepaid cards at Yahoo Auction.

Table 1 Number of unique browsers visited to Yahoo JAPAN

Date	Number of browsers
April 29, 2016	95,123,327
April 30, 2016	92,878,076
May 1, 2016	92,141,013
May 2, 2016	98,050,854
May 3, 2016	94,517,378
May 4, 2016	93,257,615
May 5, 2016	93,291,749

Table 2 Training dataset provided by Yahoo JAPAN

Label	Number of browsers	Number of logs
Postive	11	9,286
Negative	300	132,853

Table 3 Test dataset provided by Yahoo JAPAN

Label	Number of browsers	Number of logs
Postive	10	7,577
Negative	300	139,001

Table 1 shows the number of unique browser visited to Yahoo JAPAN from April 29, 2016 to May 5, 2016. Table 2 and 3 show the details of our dataset. We chose randomly from all visits except that of the fraudulent user and regarded them as negative data, to save computational resources. Positive data consisted of the web access logs of malicious users. This number was so small that training was hindered by class imbalance. The class imbalance problem is common in machine learning, where the amount of data from one class far exceed that of the other. In this situation, learning tends to fail; however, it is often the case in the security field that we cannot obtain a sufficient number of positive data examples. Therefore, it is often important that we properly model fraudulent behavior even using class-imbalanced data.

4.2 Test Environment

Our testing environment was a CentOS 7.2.1511 virtual server with a Intel Xeon 1.80GHz processor and 8 GB of memory. The programming language was JavaScript running on Node.js version 6.3.1.

4.3 Training Conditions

Our RNN structure was consisted of 153 units in the input layer, 14 units in the hidden layer, and 2 units in the output layer. Because the number of the positive data is less than the number of the negative data, the number of the units in the input layer was decided to correspond to the number of the host names in the positive data. The threshold of the value of the error function was set to 0.5. Training was to be discontinued when the value of the error function was less than the threshold. The learning coefficient was set to 0.00001. The weight of the connection was initialized by the random number according to mean 0 and standard deviation 0.001.

4.4 Value of Error Function

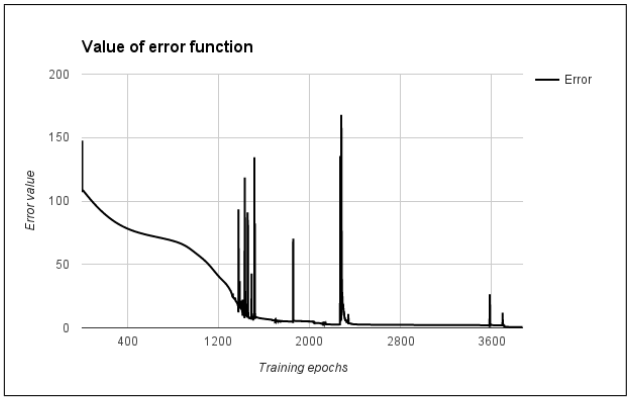

Fig. 6 Value of error function

Figure 6 shows the value of the error function as the number of times of training. There are sharp increases in places in the figure 6. These increases indicate that the failures of the gradient descent occured. However, the value of the error function finally converged after 3,875 backpropagations. The training took 17 hours and 56 minutes and 21 seconds.

4.5 Results

Table 4 shows the results of our evaluation. F-measures were calculated using the training dataset and the test dataset described in the section 4.1. We used two types of RNN, RNN without LSTM and RNN with LSTM, with sigmoid units. For this evaluation, we used not only two types of RNNs but also two types of SVM, with linear and radial basis function kernel types, respectively. The results indicate that an RNN can accurately detect malicious behavior better than an SVM. Althogh two types of RNN had the same accuracy, RNN with LSTM converged faster than RNN without LSTM.

Table 4 Evaluation Result

Algorithm	F-measure
RNN (without LSTM)	1.0000
RNN (with LSTM)	1.0000
SVM (linear kernel)	0.5967
SVM (RBF kernel)	0.5844

5. Conclusions

Our evaluation results indicate that an RNN is superior to other common machine learning techniques, such as SVMs, for learning a malicious user's behavior using web access logs.

6. Future Work

Although our approach has been proven to be effective, the factor that most contributes to this result has not yet been determined. We will investigate the relationship between patterns of malicious behavior and the internal structure of our RNN. A current hypothesis is that there are clear differences between the behaviors of legitimate and malicious users. Further research can make these differences clear using the internal parameter values of our neural network during training.

In this paper, we set as our target the user who attempts to use stolen credit cards to buy prepaid cards on an Internet auction site. We would like to generalize our approach in order to apply it to other fraudulent behaviors.

7. Acknowledgment

We would like to thank Yahoo JAPAN for providing useful datasets and computing resources.

References

[1] Anderson, R., Barton, C., Böhme, R., Clayton, R., Van Eeten, M. J., Levi, M., Moore, T., and Savage, S. Measuring the cost of cybercrime. *The economics of information security and privacy* (2013), 265–300.

[2] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *journal of machine learning research 3*, Feb (2003), 1137–1155.

[3] Burguera, I., Zurutuza, U., and Nadjm-Tehrani, S. Crowdroid: behavior-based malware detection system for android. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices* (2011), ACM, pp. 15–26.

[4] Elman, J. L. Finding structure in time. *Cognitive science 14*, 2 (1990), 179–211.

[5] Graves, A., Liwicki, M., Bunke, H., Schmidhuber, J., and Fernández, S. Unconstrained on-line handwriting recognition with recurrent neural networks. *Advances in Neural Information Processing Systems* (2008), 577–584.

[6] Graves, A., Mohamed, A.-r., and Hinton, G. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing* (2013), IEEE, pp. 6645–6649.

[7] Graves, A., and Schmidhuber, J. Offline handwriting recognition with multidimensional recurrent neural networks. *Advances in neural information processing systems* (2009), 545–552.

[8] Hochreiter, S., and Schmidhuber, J. Lstm can solve hard long time lag problems. *Advances in neural information processing systems* (1997), 473–479.

[9] Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences 79*, 8 (1982), 2554–2558.

[10] Jordan, M. I. Attractor dynamics and parallelism in a connectionist sequential network. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (1986).

[11] Losses, N. Estimating the global cost of cybercrime. *McAfee, Centre for Strategic & International Studies* (2014).

[12] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. Recurrent neural network based language model. In *Interspeech* (2010), vol. 2, p. 3.

[13] Patidar, R., and Sharma, L. Credit card fraud detection using neural network. *International Journal of Soft Computing and Engineering (IJSCE) 1*, 32-38 (2011).

[14] Rieck, K., Trinius, P., Willems, C., and Holz, T. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security 19*, 4 (2011), 639–668.

[15] Sheikhan, M., Jadidi, Z., and Farrokhi, A. Intrusion detection using reduced-size rnn based on feature grouping. *Neural Computing and Applications 21*, 6 (2012), 1185–1190.

[16] Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE 78*, 10 (1990), 1550–1560.

[17] Williams, R. J., and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation 1*, 2 (1989), 270–280.