


RESEARCH

Open Access

A deep learning framework for predicting cyber attacks rates



Xing Fang^{1*} , Maochao Xu², Shouhuai Xu³ and Peng Zhao⁴

Abstract

Like how useful weather forecasting is, the capability of forecasting or predicting cyber threats can never be overestimated. Previous investigations show that cyber attack data exhibits interesting phenomena, such as long-range dependence and high nonlinearity, which impose a particular challenge on modeling and predicting cyber attack rates. Deviating from the statistical approach that is utilized in the literature, in this paper we develop a deep learning framework by utilizing the bi-directional recurrent neural networks with long short-term memory, dubbed BRNN-LSTM. Empirical study shows that BRNN-LSTM achieves a significantly higher prediction accuracy when compared with the statistical approach.

Keywords: ARIMA, GARCH, RNN, Hybrid models, LSTM, Deep learning, BRNN-LSTM

1 Introduction

Cyber attacks have become a prevalent and severe threat against the society, including its infrastructures, economy, and citizens' privacy. According to a 2017 report by Symantec¹, cyber attacks in year 2016 include multi-million dollar virtual bank heists as well as overt attempts to disrupt the U.S. election process; according to another 2017 report by NetDiligence², the average cyber breach cost is \$394K and companies with revenues greater than \$2B suffer an average breach cost of \$3.2M.

Given the severe consequence of cyber attacks, cyber defense capability needs to be substantially improved. One approach to improving cyber defense is to forecast or predict cyber attacks, similar to how weather forecasting has benefited the society in mitigating natural hazards. The prediction capability can guide defenders to achieve cost-effective, if not optimally, allocation of defense resources [1–4]. For example, the defender may need to allocate more resources for deep packet inspection [5] to accommodate the predicted high cyber attack rate. Moreover, researchers have studied how to use a Bayesian method to predict the increase or decrease of cyber attacks [6], how to use a hidden Markov model to predict the increase or decrease of Bot agents [7], how to use a seasonal ARIMA

model to predict cyber attacks [8], how to use a FARIMA model to predict cyber attack rates when the time series data exhibits long-range dependence [1], how to use a FARIMA+GARCH model to achieve even more accurate predictions by further accommodating the extreme values exhibited by the time series data [9], how to use a marked point process to model extreme cyber attack rates while considering both magnitudes and inter-arrival times of time series [10], how to use a vine copula model to quantify the effectiveness of cyber defense early-warning mechanisms [11], and how to use a vine copula model to predict multivariate time series of cybersecurity attacks while accommodating the high-dimensional dependence between the time series [12]. We refer to two recent surveys on the use of statistical methods in cyber incident and attack detection and prediction [13, 14].

A particular kind of cyber threat data is the time series of cyber attacks observed by a cyber defense instrument known as honeypots, which passively monitor the incoming Internet connections. Such datasets exhibit rich phenomena, including long-range dependence (LRD) and highly nonlinearity [1, 9].

It is worth mentioning that the usefulness of prediction capabilities in the context of cyber defense ultimately depends on the degree of prediction accuracy, a situation similar to the usefulness of weather forecasting. This factor should be made fully aware to cyber defense practitioners. Although the prediction accuracy could be

*Correspondence: xfang13@ilstu.edu

¹School of Information Technology, Illinois State University, Normal 61761, IL, USA

Full list of author information is available at the end of the article

assured by leveraging large amounts of data, which is indeed true to the case of weather forecasting, the collection of large amounts of cyber attack data may be challenging. Nevertheless, understanding the usefulness of prediction capabilities in the context of cyber security is a problem of high importance but has yet to be thoroughly investigated.

1.1 Our contributions

The contribution of the present paper is in two-fold. First, we propose a novel bi-directional recurrent neural networks with long short-term memory framework, or BRNN-LSTM for short, to accommodate the statistical properties exhibited by cyber attack rate time series data. The framework gives users the flexibility in choosing the number of LSTM layers that are incorporated into the BRNN structure. Second, we use real-world cyber attack rate datasets to show that BRNN-LSTM can achieve a substantially higher prediction accuracy than statistical prediction models, including the one proposed in literature [9] and the ones that are studied in the present paper for comparison purposes.

1.2 Related work

Statistical methods have been widely used in the context of data-driven cyber security research, such as intrusion detection [15–18]. However, deep learning has not received the due amount of attention in the context of cyber security [13, 14]. This is true despite the fact that deep learning has been tremendously successful in other application domains [19–21] and has started to be employed in the cyber security domains, including adversarial malware detection [22, 23] and vulnerability detection [24, 25].

In the context of vulnerability detection, supervised machine learning methods including logistic regression, neural network, and random forest, have been proposed for this purpose [26, 27]. These models are trained using large-scale vulnerability data. However, unlike deep learning models that can directly work on raw data, those models require the data to be preprocessed to extract features. There are also other approaches to detecting vulnerabilities. For example, an architectural approach to pinpointing memory-based vulnerabilities has been proposed in [28], which consists of an online attack detector and an offline vulnerability locator that are linked by a record and replay mechanism. Specifically, it records the execution history of a program and simultaneously monitors its execution for attacks. If an attack is detected by the online detector, the execution history is replayed by the offline locator to locate the vulnerability that is being exploited. For more discussions on the vulnerability detection, please refer to [24, 25, 27, 28], and the references therein.

In the context of time series analytics, various statistical approaches have been developed. For example, ARIMA, Holt-Winters, and GARCH models are among the most popular statistical approaches for analyzing time series data [1, 8, 9, 29]. Other statistical models, such as Gaussian mixture models, hidden Markov models, and state space models have been developed to analyze time series data with uncertainties and/or some unobservable factors [17, 30]. Recently, it was discovered that deep learning is very efficient in time series prediction. For example, deep learning has been employed to predict financial data, which contains some noise and volatility [21]. In the context of transportation application, deep learning has been used to predict passenger demands for on-demand ride service [31]. In particular, it is discovered that deep learning can achieve a higher accuracy than statistical time series models (e.g., ARMA and Holt-Winters models) in predicting transportation traffic [32–34]. It is further argued in [32] that a particular class of deep learning models, known as feed-forward neural networks, are the best predictors when taking into account both prediction precision and model complexity. In [34], the prediction performances of the deep learning approach and of the statistical ARIMA approach are compared against each other. It is shown that the deep learning approach can significantly (more than 80%) reduce the error rate when compared with the ARIMA models.

The rest of the paper is organized as follows. In the “Preliminaries” section, we review some concepts of deep learning that are related to the deep learning framework we will propose in this paper. In the “Framework” section, we present the framework we propose for predicting cyber attack rates. In the “Empirical study” section, we present our experiments on applying the framework to a dataset of cyber attack rates and compare the resulting prediction accuracy with the accuracy of the statistical approach reported in the literature. In the “Conclusion” section, we conclude the present paper with future research directions.

In order to improve the readability of the paper, we summarize the main notations that are used in the present paper in Table 1:

2 Preliminaries

In this section, we review three deep learning concepts that are related to the present work: recurrent neural network (RNN), bi-directional RNN, and long short-term memory (LSTM).

2.1 RNN

Figure 1 highlights the standard RNN structure, which updates its hidden layers according to the information received from the input layer and the activation from the previous forward propagation. When compared with

Table 1 Summary of notations

W_x	Weight matrix connecting the input layer and the hidden layer
W_h	Weight matrix connecting two consecutive hidden states
W_y	Weight matrix connecting the hidden state and the output layer
b_h	Bias vector in hidden layer
b_y	Bias vector in output layer
h_t	Hidden state at time t
$\sigma(\cdot)$	Activation function
\mathbf{x}_t	Input at time t
y_t	Real output at time t
\hat{y}_t	Predicted output at time t
J	Objective function

feed-forward neural networks, RNN can accommodate the temporal information embedded into the sequence of input data (see, e.g., [35, 36]). Intuitively, this explains why RNN is suitable for natural language processing and time series analysis (see, e.g., [36–39]). This observation motivates us to leverage RNN as a starting point in designing our framework that will be presented later.

As highlighted in Fig. 1, the computing process at each time step of RNN is

$$h_t = \sigma(W_x \cdot x_t + W_h \cdot h_{t-1} + b_h),$$

where $W_x \in \mathbf{R}^{m \times n}$ is the weight matrix connecting the input layer and the hidden layer with m being the size of the input and n being the size of the hidden layer, $W_h \in \mathbf{R}^{n \times n}$ is the weight matrix between two consecutive hidden states h_{t-1} and h_t , b_h is the bias vector of the hidden layer, and σ is the activation function to generate the hidden state. As a result, the network output can be described by

$$y_t = \sigma(W_y \cdot h_t + b_y),$$

where $W_y \in \mathbf{R}^n$ is the weight connecting the hidden layer and the output layer, b_y is the bias vector of the output layer, and σ is the activation function of the output layer.

2.2 Bi-directional RNN

A *uni-directional* RNN is a RNN that only takes one sequence as the input. A uni-directional RNN cannot take full advantage of the input data in the sense that it only learns information from the “past.” In order to overcome this issue, the concept of *bi-directional* RNN is introduced to make a RNN learn from both the past and the future [40]. Technically speaking, a bi-directional RNN is essentially two uni-directional RNNs that are combined together, where one learns from the past and the other learns from the “future”; the results of the two uni-directional RNNs are merged together to compute a final output.

2.3 LSTM

The training process of RNNs can suffer from the gradient vanishing/exploding problem [41], which can be alleviated by another RNN structure known as LSTM [42]. LSTM is composed of units called *memory blocks*, each of which contains some *memory cells* with self-connections, which store (or remember) the temporal state of the network, and some special multiplicative units called *gates*. Each memory block contains an *input gate*, which controls the flow of input activations into the memory cell; an *output gate*, which controls the output flow of cell activations into the rest of the network; and a *forget gate*.

As highlighted in Fig. 2, the activation at step t , namely, h_t , is computed based on four pieces of gate input, namely, the information gate i_t , the forget gate f_t , the output gate o_t , and the cell gate c_t [43]. Specifically, the information gate input at step t is

$$i_t = \sigma(U_i \cdot h_{t-1} + W_i \cdot \mathbf{x}_t + b_i),$$

where $\sigma(\cdot)$ is a sigmoid activation function, b_i is the bias, \mathbf{x}_t is the input vector at step t , and W_i and U_i are weight matrices. The forget gate input and the output gate input are respectively computed as

$$\begin{aligned} f_t &= \sigma(U_f \cdot h_{t-1} + W_f \cdot \mathbf{x}_t + b_f), \\ o_t &= \sigma(U_o \cdot h_{t-1} + W_o \cdot \mathbf{x}_t + b_o), \end{aligned}$$

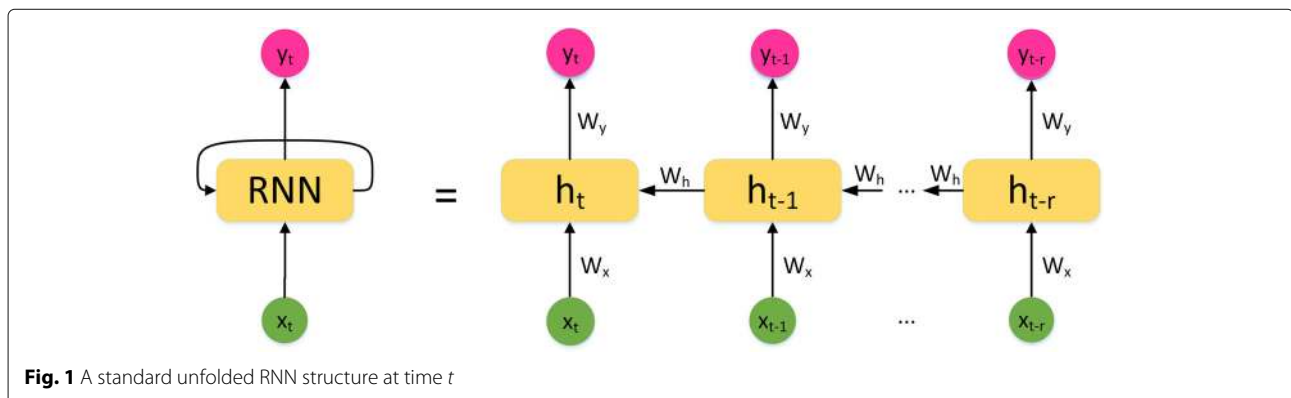
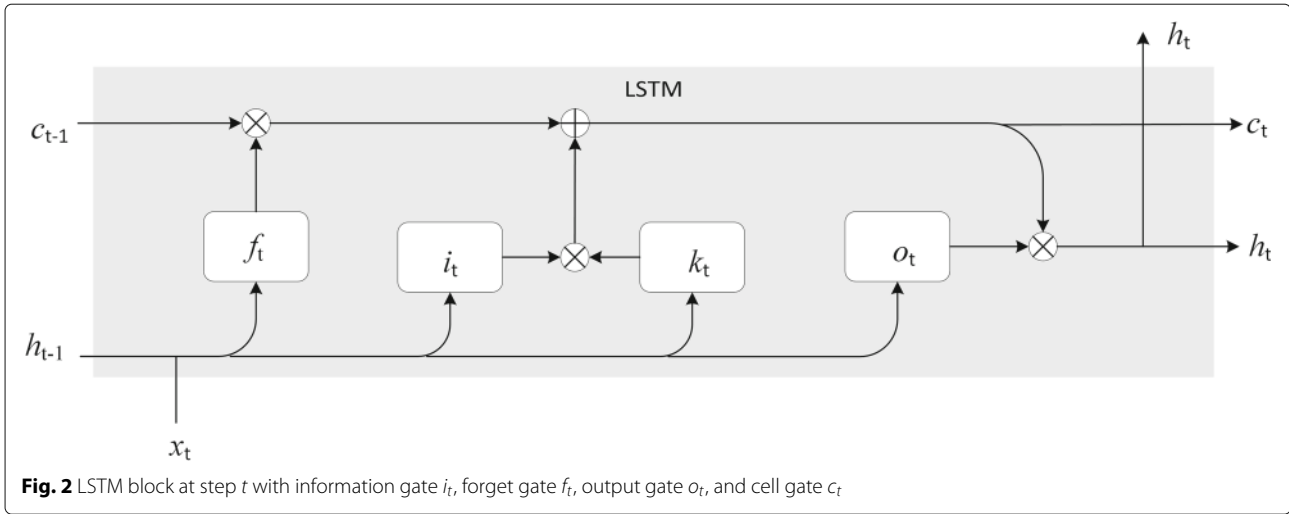


Fig. 1 A standard unfolded RNN structure at time t



where U_f , U_o , W_f , and W_o are weight matrices, and b_f and b_o are biases. The cell gate input is computed as

$$c_t = f_t \cdot c_{t-1} + i_t \cdot k_t \text{ with } k_t = \tanh(U_k \cdot h_{t-1} + W_k \cdot \mathbf{x}_t + b_k),$$

where \tanh is the hyperbolic tangent function, U_k and W_k are weights, and b_k is bias. The activation at step t is computed as

$$h_t = o_t \cdot \tanh(c_t).$$

Intuitively, the key component of LSTM is the *cell state*, which flows throughout the network. Given input h_{t-1} and \mathbf{x}_t , the *forget gate* f_t decides to throw away what information from the previous cell state c_{t-1} . The forget gate f_t takes h_{t-1} and \mathbf{x}_t as input and uses the sigmoid activation function $\sigma(\cdot)$ to generate a number between 0 and 1 for each value in cell state c_{t-1} . The information gate i_t determines what new information in the current cell state c_t to be stored, via two steps: a set of candidate values are computed by k_t based on the current input; the information gate i_t then uses $\sigma(\cdot)$ to decide which candidate values will be stored in c_t . The cell gate will then compute c_t . Finally, h_t is computed based on c_t and o_t , where the latter is the information from the output gate.

3 The bi-directional RNN with LSTM framework

The framework we propose for predicting cyber attack rates is called bi-directional RNN with LSTM or BRNN-LSTM for short, which incorporates some LSTM layers into a bi-directional RNN. BRNN-LSTM has three components: an input layer, a number of hidden layers, and an output layer, where each hidden layer is replaced with a LSTM cell. The same sequential input, denoted by $\mathbf{x}_t = \{x_0, \dots, x_t\}$, is passed to the two states of the LSTM layers, the forward state, and the backward state. There is no connection in between the two states. The outputs from

the two states are then combined together to predict a target value at each step. Figure 3 highlights the structure of BRNN-LSTM with three LSTM layers.

For training a BRNN-LSTM model, we propose using the following objective function:

$$J = \frac{1}{2m} \cdot \sum_{i=1}^m (\hat{y}_i - y_i)^2 + \frac{\lambda}{2} (\|\mathbf{W}\|_2^2 + \|\mathbf{U}\|_2^2), \quad (1)$$

where m is the size of the input, \hat{y}_i and y_i are respectively the output of network and the observed values at step i , \mathbf{W} and \mathbf{U} are weight matrices, $\mathbf{W} = \{W_f, W_i, W_k, W_o\}$, $\mathbf{U} = \{U_f, U_i, U_k, U_o\}$, $\|\cdot\|_2^2$ represents the squared L_2 norm of weight matrices, and λ is a user-defined penalty parameter. Note that the second term in Eq. (1) is the penalty term for avoiding overfitting. The optimization is defined as

$$\Theta^* = \arg \min_{\Theta} J,$$

where $\Theta = (\mathbf{W}, \mathbf{U})$ are model parameters and can be solved by using the gradient descent method [42, 44].

4 Empirical study

4.1 Accuracy metrics

Let (y_1, \dots, y_N) be observed values and $(\hat{y}_1, \dots, \hat{y}_N)$ be the predicted values. In order to evaluate the accuracy of the BRNN-LSTM framework, we propose using the following widely used metrics [1, 9, 45].

- Mean square error (MSE):

$$\text{MSE} = \sum_{i=1}^N (y_i - \hat{y}_i)^2 / N.$$
- Mean absolute deviation (MAD):

$$\text{MAD} = \sum_{i=1}^N |y_i - \hat{y}_i| / N.$$
- Percent mean absolute deviation (PMAD):

$$\text{PMAD} = \sum_{i=1}^N |y_i - \hat{y}_i| / \sum_{i=1}^N |y_i|.$$
- Mean absolute percentage error (MAPE):

$$\text{MAPE} = \sum_{i=1}^N |(y_i - \hat{y}_i) / y_i| / N.$$

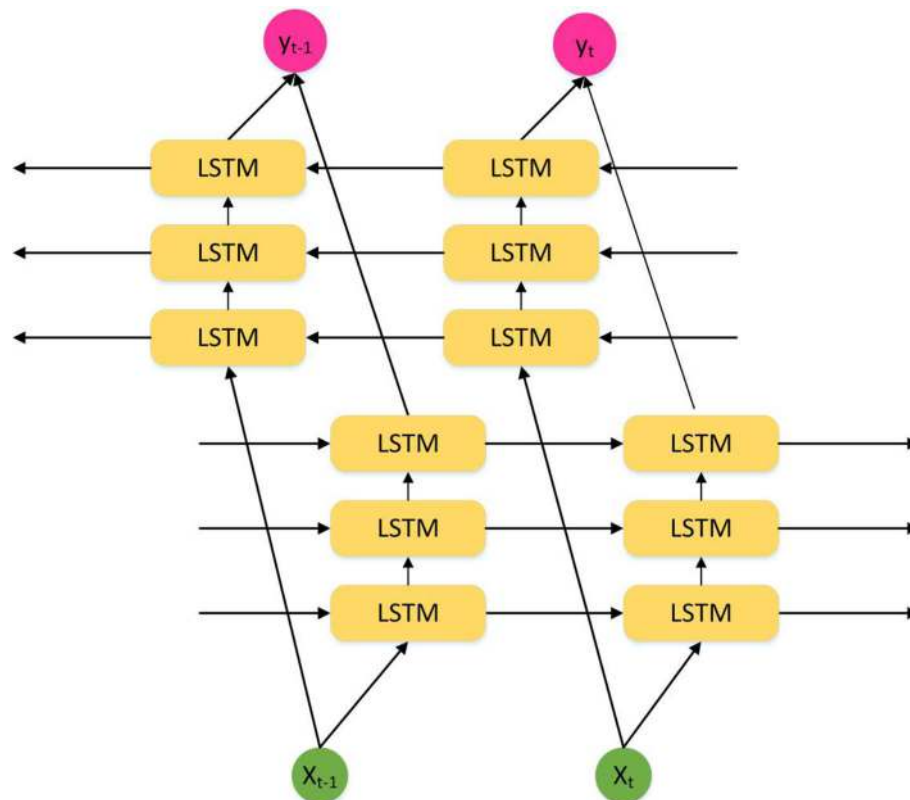


Fig. 3 BRNN-LSTM with three LSTM layers

4.2 Data collection

The dataset we analyze is the same as the dataset analyzed in [1]. The dataset was collected by a low-interaction honeypot consisting of 166 consecutive IP addresses during five periods of time in the interval between year 2010 and year 2011. These five periods of time are respectively 1,123, 421, 1,375, 528, and 1920 h, each of which is represented by a separate dataset. The honeypot runs the following four honeypot programs: Dionaea³, Mwcollector⁴, Amun⁵, and Nepenthes [46], which run some vulnerable services such as SMB (with Microsoft Windows Server Service Buffer Overflow vulnerability MS06040 and Workstation Service Vulnerability MS06070), NetBIOS, HTTP, MySQL and SSH. A honeypot computer runs multiple honeypot programs, each of which monitors (i.e., is associated to) one IP address. A dedicated computer collects the raw network traffic coming to the honeypot as pcap files. Honeypot-captured data are treated as cyber attacks because no legitimate services are associated to the honeypot computers. We refer to [1] for more details about the honeypot instrument.

4.3 Data preprocessing

As in [1] and many analyses, we treat flows (rather than packets) as attacks, while noting that flows can be based

on the TCP or UDP protocol. A TCP flow is uniquely identified by an attacker's IP address, the port used by the attacker to wage the attack, a victim IP address (belonging to the honeypot), and the port of the victim IP address under attack. An unfinished TCP handshake is also treated as a flow or attack because the success may be attributed to the fact that the connection is dropped because the port in question is busy. Also as in [1], the preprocessing contains the following steps. First, we disregard the cyber attacks that are waged against the non-production (i.e., unassigned) ports (i.e., any ports that are not associated with the honeypot programs) because these TCP connections are often dropped. Since low-interaction honeypot programs do not collect adequate traffic information that would allow us to determine specific attacks, we only consider the attack rate or the number of attacks (rather than specific types of attacks). Second, the following two widely used parameters [47] are also used to preprocess network traffic flows not ending with the FIN flag (meaning that these flows are terminated unsafely) or the RST flag (meaning that these flows are terminated unnaturally): 60 s for the *flow timeout time* (meaning that an attack or flow expires after being idle for 60 s) and 300 s for the *flow lifetime* (meaning that an attack or flow does not span over 5 min or 300 s).

Algorithm 1 Algorithm for computing fitted values.

INPUT: Historical time series data $\{(t, y_t) | t = 1, \dots, m\}$; iteration $b = 10,000$; penalty parameter $\lambda = .001$.

```

1: for  $r \in \{20, 30, 40\}$  do
2:   Split the data set into mini-batch of size  $r$ 
3:   for  $l \in \{2, 3, 4, 5\}$  do
4:     Randomly initialize a  $l$ -layer BRNN-LSTM with
       parameters saved in  $\Theta$ 
5:      $j \leftarrow 0$ 
6:     while  $j \leq b$  do
7:       Compute  $J$  in Eq. (1) by performing forward
         propagation
8:       Update  $\Theta$  using the Adam optimizer [44]
9:        $j \leftarrow j + 1$ 
10:    end while
11:    for each data point at  $t$  do
12:      Compute  $\hat{y}_t$  by performing forward
        propagation
13:      Fitted value  $\leftarrow \hat{y}_t$ 
14:    end for
15:    return Fitted values for combination  $(r, l)$ .
16:  end for
17: end for

```

OUTPUT: Fitted values for various combinations of (r, l) 's.

For each period or dataset, the data is represented by $\{(t, x_t)\}$ for $t = 0, 1, 2, \dots$, where x_t is the number of attacks (i.e., attack rate) that are observed by the honeypot at time t . Unlike [1], we further preprocess the derived attack rate time series by normalizing attack rates into interval $(0, 1]$. Then, small data batches (periods) are selected based on a pre-defined mini-batch size. For prediction purposes, we split each time series into an in-sample part (for model training) and an out-of-sample part (for prediction). As in [1], we set the last 120 h of each period as the out-of-sample part for evaluating prediction accuracy.

4.4 Model training and selection

In the training process, we use the mini-batch gradient descent method to compute the minimum of the objective function, which is described in Eq. (1). We use 10,000 iterations to train a network and set the penalty parameter $\lambda = .001$ because other parameters do not lead to any significantly better result. For each dataset, we use Algorithm 1 to compute the fitted values with varying model parameters. We select the model that achieves the minimum MSE.

Table 2 describes the selected model and MSE for each dataset. We observe that the selected model for different

datasets may use different batch size r and different number l of LSTM layers. For datasets I, IV, and V, the selected batch size is 20; for datasets II and III, the selected batch size is respectively 30 and 40. For the number of LSTM layers, datasets I and IV prefer to 4 layers; datasets II and V prefer to 2 layers; and period IV prefers to 3 layers.

Figure 4 plots the fitting of the selected model corresponding to each dataset. We observe that the selected models have satisfactory fitting accuracy. In particular, the extreme values are fitted well in every dataset.

4.5 Prediction accuracy

We use Algorithm 2 to predict cyber attack rates corresponding to the out-of-samples, which allow us to calculate the prediction accuracy.

Algorithm 2 Algorithm for predicting cyber attack rates

INPUT: Historical time series data with in-sample set $\{(t, y_t) | t = 1, \dots, m\}$ and out-of-sample set $\{(t, y_s) | s = m + 1, \dots, n\}$; iteration $b = 10,000$; penalty parameter $\lambda = .001$; (r, l) selected in model training.

```

1: Split the in-sample set into mini-batches of size  $r$ 
2: Randomly initialize a  $l$ -layer BRNN-LSTM, with all
   the parameters saved in  $\Theta$ 
3:  $j \leftarrow 0$ 
4: while  $j < b$  iteration do
5:   for Each mini-batch from the in-sample set do
6:     Compute  $J$  by performing forward propagation
7:     Update  $\Theta$  using the Adam optimizer ([44])
8:   end for
9:    $j \leftarrow j + 1$ 
10: end while
11: Predictions  $\leftarrow \emptyset$ 
12: for Each data point,  $s$ , in the out-of-sample set do
13:   Compute  $\hat{y}_s$  by performing forward propagation
14:   Predictions  $\leftarrow \hat{y}_s$ 
15: end for
16: return Predicted values

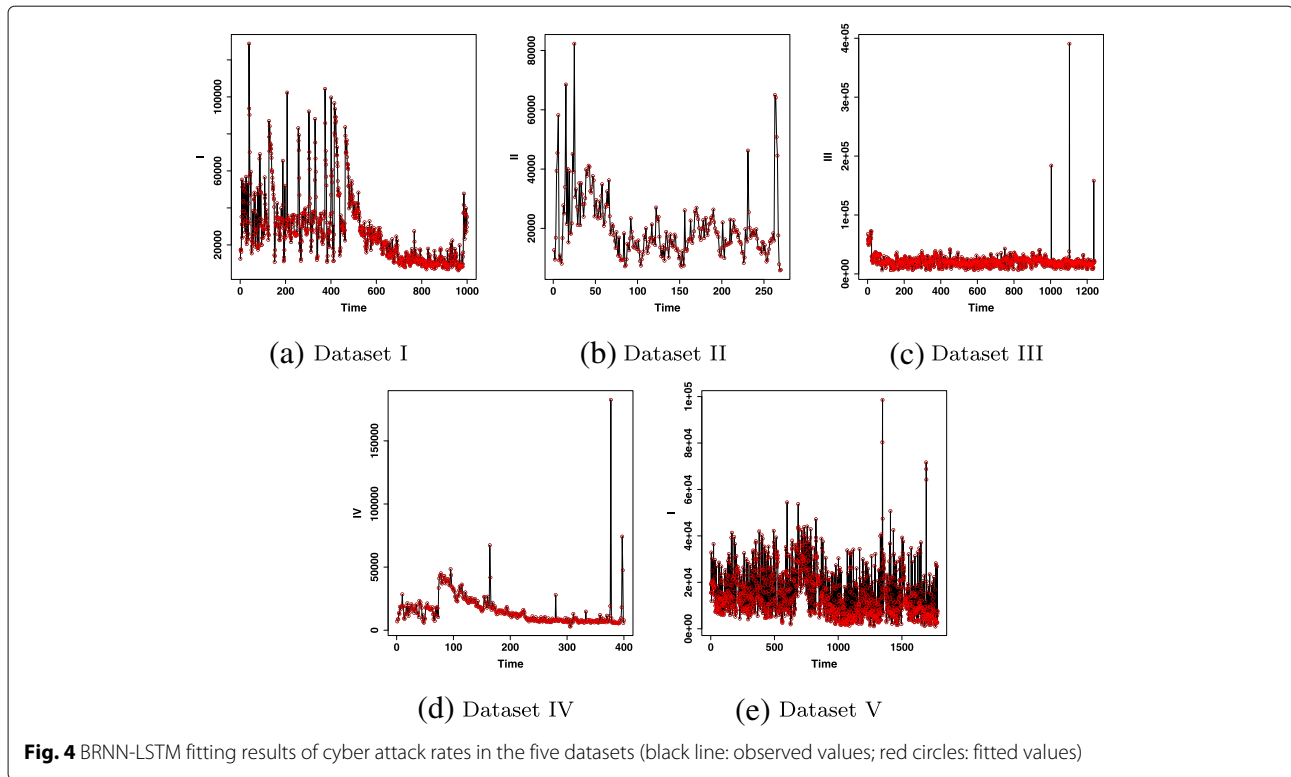
```

OUTPUT: Predicted values.

Table 3 describes the prediction results in terms of the accuracy metrics mentioned above. Based on metrics PMAD and MAPE, BRNN-LSTM achieves a remarkable prediction accuracy for datasets I, II, III, and V because prediction errors are less than 5%. However, for dataset IV,

Table 2 Parameters (r, l) of selected model and MSE for each dataset

Dataset	I	II	III	IV	V
r	20	30	40	20	20
l	4	2	4	3	2
MSE	4051.685	935.0724	251061.7	2898.278	9132.973



the prediction accuracy in metric PMAD is around 17% and in metric MAPE is around 27%. Fortunately, BRNN-LSTM can be easily calibrated to improve its prediction accuracy via a rolling approach as follows. For period IV, we re-estimate model parameters in Θ via Algorithm 1 after observing 20 more data points; the corresponding prediction accuracy, indicated by “IV*” in Table 3, is much better than the original prediction accuracy. For example, the rolling approach reduces the PMAD metric to 10% and reduces the MAPE metric to 13%.

Figure 5 plots the prediction results. We observe that predicted values match observed values well, but some observed values that are still missed by BRNN-LSTM. For example, for dataset III, the extreme value is missed and some observed values are over-predicted. Nevertheless, we conclude that the prediction accuracy is satisfactory.

4.6 Model comparisons

In order to further evaluate the prediction accuracy of the proposed framework, we now compare it with other popular models.

4.6.1 ARIMA

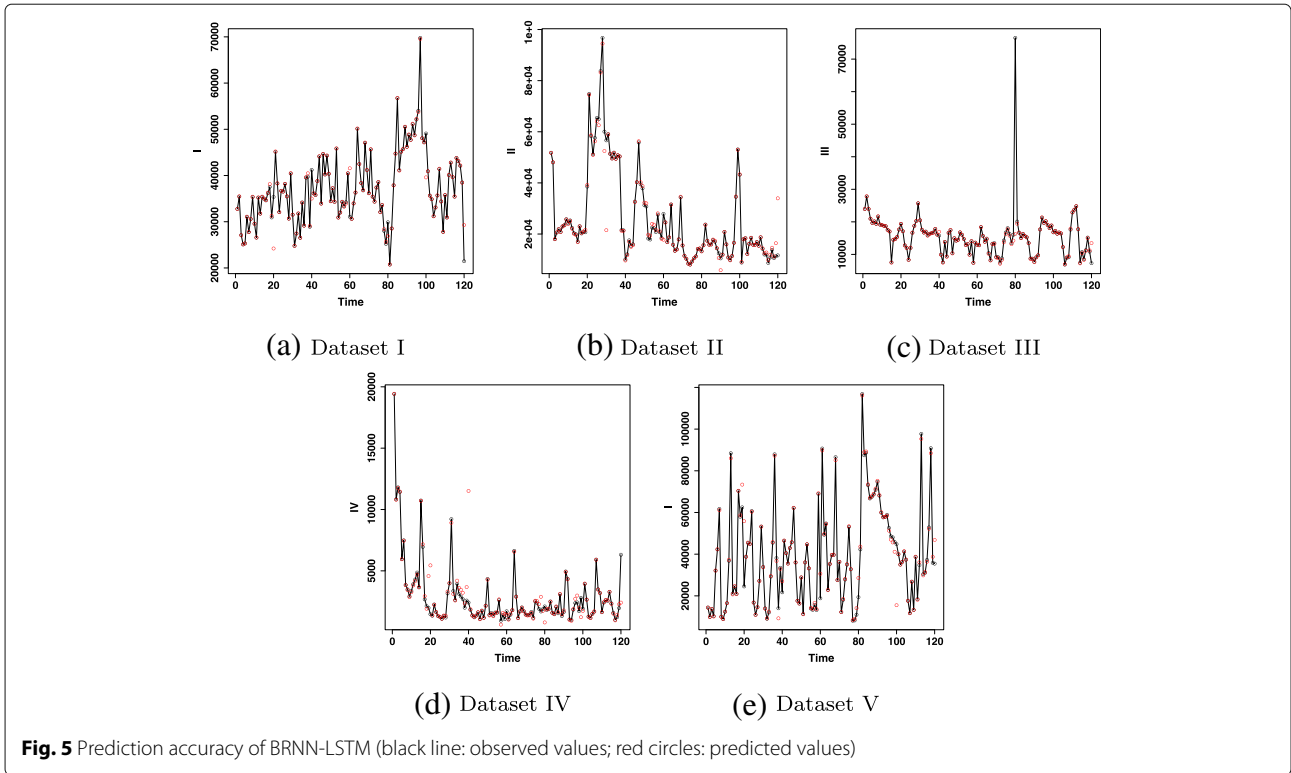
The first model we consider (as a benchmark) is the AutoRegressive Integrated Moving Average or ARIMA (p, d, q) , which is perhaps the most well-known model in time series analysis [29, 30]. The ARIMA model is described as

$$\phi(B)(1 - B)^d Y_t = \theta(B)e_t,$$

where B is the backshift operator, and $\phi(B)$ and $\theta(B)$ are respectively the AR and MA characteristic polynomials evaluated at B . In order to select the ARIMA model for

Table 3 Parameters of selected models and prediction accuracy metrics of these selected models, where IV* indicates the rolling approach for dataset IV

Dataset	Test	r	l	MSE	MAD	PMAD	MAPE
I	120	20	4	3,628,266	463.2715	.01243741	.01387808
II	120	30	2	16,497,941	1036.6035	.04012863	.04819186
III	120	40	4	30,637,599	675.7551	.04299127	.02304677
IV	120	20	3	2,165,707	508.3557	.1658243	.26563720
IV*	120	20	3	1,085,361	297.3440	.1034426	.13385770
V	120	20	2	20,415,119	1396.7624	.03564086	.04787385



prediction purpose, we use the AIC criterion while allowing the orders of p and q to vary from 0 to 5 and d to vary from 0 to 2.

4.6.2 ARMA+GARCH

The second model we consider further incorporates the Generalized AutoRegressive Conditional Heteroscedastic or GARCH model, which is widely used in financial time series applications. We use GARCH(1, 1) to model the conditional variance and the ARMA model to accommodate the conditional mean. This leads to the following ARMA+GARCH model:

$$Y_t = E(Y_t | \mathfrak{F}_{t-1}) + \epsilon_t,$$

where $E(\cdot)$ is the conditional expectation function, \mathfrak{F}_{t-1} is the historic information up to time $t - 1$, and ϵ_t is the innovation of the time series. Since the mean part is modeled as ARMA(p, q), the model can be rewritten as

$$Y_t = \mu + \sum_{k=1}^p \phi_k Y_{t-k} + \sum_{l=1}^q \theta_l \epsilon_{t-l} + \epsilon_t, \tag{2}$$

where $\epsilon_t = \sigma_t Z_t$ with Z_t being i.i.d. innovations. For the standard GARCH(1, 1) model, we have

$$\sigma_t^2 = w + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2, \tag{3}$$

where σ_t^2 is the conditional variance and w is the intercept. After some preliminary analysis, we set the order

of ARMA to (1, 1) as a higher order does not provide significant better predictions.

4.6.3 Hybrid model

The third model we consider is based on the recently developed hybrid approach, which is a two-step procedure [48, 49]. The hybrid model first extracts the linear relationship using an ARIMA model, and then uses a nonlinear approach to determine the nonlinear relationship. The nonlinear step can be considered as a prediction on the error term. The resulting hybrid model is written as

$$Y_t = L_t + N_t,$$

where L_t is the linear part and N_t is the nonlinear part. Since L_t is modeled by an ARIMA model, the residuals at time t are

$$e_t = Y_t - \hat{Y}_t,$$

where \hat{Y}_t is the fitted value. The residuals are modeled by a nonlinear model, which utilizes the lag information. We consider the following three types of hybrid models:

- H1 : $N_t = f(e_{t-1}, e_{t-2}, \dots, e_{t-n}) + \epsilon_t,$
- H2 : $N_t = f(e_{t-1}, e_{t-2}, \dots, e_{t-n}, y_{t-1}, y_{t-2}, \dots, y_{t-m}) + \epsilon_t,$
- H3 : $N_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-n}) + \epsilon_t,$

where ϵ_t is the random error at time t and f is a nonlinear function. For nonlinear function f , we consider the following three popular machine learning approaches

[50]: random Forest or RF [49], support vector machine or SVM [51], and artificial neural network or ANN [48, 52].

In order to achieve the best prediction accuracy, we examine a number of models. For the linear part of ARIMA(p, d, q), we use the AIC criterion to select models in the training process, where p and d vary from 0 to 5 and q varies from 0 to 1. For the nonlinear model, we vary the lag parameter from 1 to 12. All of the models are trained by using 10-folder validation. For RF, we set the number of trees to 1000; for SVM, we consider the following kernel functions: linear, polynomial, radial basis, and sigmoid; for ANN, we set the number of hidden layers to one while varying the number of hidden nodes from 1 to 10.

4.6.4 Comparison

We select the highest prediction accuracy in terms of the MSE metric derived from the predicted values and the out-of-sample data. For dataset I, the best prediction model is ARIMA(2,1,1)+ANN+H3 with the number of lags being 5 and 8 hidden nodes. For dataset II, the best prediction model is ARIMA(3,1,1)+“linear SVM”+H2 with the number of lags being 6. For dataset III, the best prediction model is ARIMA(3,0,1)+“radial SVM”+H3 with the number of lags being 8. For dataset IV, the best prediction model is ARIMA(0,1,2)+“radial SVM”+H1 with the number of lags being 4. For dataset V, the best prediction model is ARIMA+“radial SVM”+H3 with the number of lags being 7.

Table 4 summarizes the one-step ahead rolling prediction accuracy. Considering the MSE metric, we observe that the ARIMA model has the worst prediction accuracy for datasets I–IV, and the hybrid model outperforms the ARMA+GARCH model for every dataset; we also observe that the ARIMA model has the smallest MSE for dataset V. Considering the MAD metric, we observe that the hybrid model outperforms the other two models for datasets I, III, and IV, but the ARMA+GARCH model outperforms the other two models for dataset II; we also observe that the ARIMA model has the smallest MAD for dataset IV. Considering metrics PMAD and MAPE, we observe that the hybrid model outperforms the other two models for datasets I, III, IV, and V, and the ARMA+GARCH model is slightly better than the hybrid model for dataset II; we also observe that all of the models have the worst prediction accuracy for datasets IV and V, which coincides with the conclusion drawn in [9], namely, that the PMADs of one-step ahead rolling prediction of the FARIMA+GARCH model are respectively 0.138, 0.121, 0.140, 0.339, and 0.378 for the five datasets. By comparing Tables 3 and 4, we draw:

Insight 1 *The BRNN+LSTM framework achieves a higher prediction accuracy than the FARIMA+GARCH*

Table 4 Prediction accuracy of the selected model with respect to each dataset

Dataset	MSE	MAD	PMAD	MAPE
ARIMA				
I	40,054,811	5,038.95	0.1352803	0.1378065
II	100,487,103	6,763.351	0.2618205	0.314159
III	47,486,461	3,478.307	0.2212886	0.2573687
IV	17,002,355	2,353.409	0.8187241	0.8372556
V	456,948,359	15,919.9	0.4062245	0.5932768
ARMA+GARCH				
I	38,077,842	4908.317	0.1317732	0.1361043
II	93,164,156	5,861.041	0.2268906	0.2530479
III	56,736,538	3431.358	0.2183016	0.2395564
IV	3,837,969	1,356.005	0.4717387	0.5876807
V	553,535,870	16,671.04	0.4253909	0.5267857
Hybrid				
I	36,177,293.39	4,652.507998	0.124905523	0.127347065
II	93017462.9	6169.871649	0.238845915	0.281375049
III	39,425,972.04	2,807.162152	0.178590549	0.206457204
IV	3,162,758.321	1,063.447725	0.369961347	0.384547602
V	493,400,639.5	16,787.20604	0.385329179	0.516025677

model proposed in [9] and the ARIMA, ARIMA+GARCH, and hybrid models considered above.

5 Conclusion

We proposed a BRNN-LSTM framework for predicting cyber attack rates. The framework can accommodate complex phenomena exhibited by datasets, including long-range dependence and highly nonlinearity. Using five real-world datasets, we showed that the framework significantly outperforms the other prediction approaches in terms of prediction accuracy, which confirms that LSTM cells can indeed accommodate the long memory behavior of cyber attack rates. From these five datasets, we found that only dataset IV requires to re-training the model in order to achieve a better prediction accuracy. We compared the prediction accuracy of BRNN-LSTM and other prediction approaches, which use rolling predictions (i.e., re-building the prediction model after observing a new value). We hope the present work will inspire more research in deploying deep learning to prediction tasks in the cybersecurity domain.

Endnotes

¹ <https://www.symantec.com/security-center/threat-report>

² <https://netdiligence.com/portfolio/cyber-claims-study/>

³ <http://dionaea.carnivore.it/>

⁴ <https://alliance.mwcollect.org/>

⁵ <http://amunhoney.sourceforge.net/>

Abbreviations

ARIMA: Autoregressive integrated moving average; BRNN: Bi-directional recurrent neural network; GARCH: Generalized autoregressive conditional heteroskedasticity; LSTM: Long short-term memory; RNN: Recurrent neural network

Acknowledgements

Not applicable.

Funding

Not applicable.

Availability of data and materials

Data used in this work is not suitable for public use. The source code used in the present paper is available at <https://github.com/xingfang912/time-series-analysis>

Competing interests

The authors declare that they have no competing interests.

Authors' contributions

XF constructed the deep learning framework and performed the deep learning experiments. MX and PZ performed the experiments on the statistical models. SX drafted the manuscript. All authors reviewed the draft. All authors read and approved the final manuscript.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Author details

¹School of Information Technology, Illinois State University, Normal 61761, IL, USA. ²Department of Mathematics, Illinois State University, Normal 61761, IL, USA. ³Department of Computer Science, University of Texas at San Antonio, San Antonio 78249, TX, USA. ⁴Department of Computer Science, Jiangsu Normal University, Xuzhou 221110, China.

Received: 29 November 2018 Accepted: 3 May 2019

Published online: 22 May 2019

References

- Z. Zhan, M. Xu, S. Xu, Characterizing honeypot-captured cyber attacks: Statistical framework and case study. *IEEE Trans. Inf. Forensic Secur.* **8**(11), 1775–1789 (2013)
- E. Gandotra, D. Bansal, S. Sofat, Computational techniques for predicting cyber threats. *Intell. Comput. Commun. Devices Proc ICCD 2014*, **1**, 247 (2014)
- S. Xu, in *Proc. Symposium on the Science of Security (HotSoS'14)*. Cybersecurity dynamics (ACM, Raleigh, 2014), pp. 14–1142
- S. Xu, in *Proactive and Dynamic Network Defense*, ed. by Z. Lu, C. Wang. Cybersecurity dynamics: A foundation for the science of cybersecurity (Springer International Publishing, New York City, 2018)
- L. D. Carli, R. Sommer, S. Jha, in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. Beyond pattern matching: A concurrency model for stateful deep packet inspection (ACM, Scottsdale, 2014), pp. 1378–1390
- C. Ishida, Y. Arakawa, I. Sasase, K. Takemori, in *Proceedings of PACRIM. 2005 IEEE Pacific Rim Conference on Communications, Computers and signal Processing, August 24-26*. Forecast techniques for predicting increase or decrease of attacks using bayesian inference (IEEE, Victoria, 2005), pp. 450–453
- D. H. Kim, T. Lee, S.-O. D. Jung, H. P. In, H. J. Lee, in *Information Assurance and Security, 2007. IAS 2007. Third International Symposium On*. Cyber threat trend analysis model using HMM (IEEE, Manchester, 2007), pp. 177–182
- Z. Yong, T. Xiaobin, X. Hongsheng, in *Computational Intelligence and Security, 2007 International Conference On*. A novel approach to network security situation awareness based on multi-perspective analysis (IEEE, Harbin, 2007), pp. 768–772
- Z. Zhan, M. Xu, S. Xu, Predicting cyber attack rates with extreme values. *IEEE Trans. Inf. Forensic Secur.* **10**(8), 1666–1677 (2015)
- C. Peng, M. Xu, S. Xu, T. Hu, Modeling and predicting extreme cyber attack rates via marked point processes. *J. Appl. Stat.* **44**(14), 2534–2563 (2017)
- M. Xu, L. Hua, S. Xu, A vine copula model for predicting the effectiveness of cyber defense early-warning. *Technometrics*. **59**(4), 508–520 (2017)
- C. Peng, M. Xu, S. Xu, T. Hu, Modeling multivariate cybersecurity risks. *J. Appl. Stat.* **45**(15), 2718–2740 (2018)
- N. Sun, J. Zhang, P. Rimba, S. Gao, Y. Xiang, L. Y. Zhang, Data-driven cybersecurity incident prediction: A survey. *IEEE Commun. Surv. Tutor.*, 1–1 (2018). <https://doi.org/10.1109/COMST.2018.2885561>
- M. Husák, J. Komárková, E. Bou-Harb, P. Čeleda, Survey of attack projection, prediction, and forecasting in cyber security. *IEEE Commun. Surv. Tutor.* **21**(1), 640–660 (2019)
- D. E. Denning, An intrusion-detection model. *IEEE Trans. Softw. Eng.* **SE-13**(2), 222–232 (1987)
- M. Markou, S. Singh, Novelty detection: a review part 1: statistical approaches. *Sig. Process.* **83**(12), 2481–2497 (2003)
- V. Chandola, A. Banerjee, V. Kumar, Anomaly detection: a survey. *ACM Comput. Surv. (CSUR)*. **41**(3), 15 (2009)
- J. Neil, C. Hash, A. Brugh, M. Fisk, C. B. Storlie, Scan statistics for the online detection of locally anomalous subgraphs. *Technometrics*. **55**(4), 403–414 (2013)
- L. Deng, D. Yu, et al., Deep learning: methods and applications. *Found. Trends® Sig. Process.* **7**(3–4), 197–387 (2014)
- M. Långkvist, L. Karlsson, A. Loutfi, A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recogn. Lett.* **42**, 11–24 (2014)
- R. C. Cavalcante, R. C. Brasileiro, V. L. Souza, J. P. Nobrega, A. L. Oliveira, Computational intelligence and financial markets: A survey and future directions. *Expert Syst. Appl.* **55**, 194–211 (2016)
- D. Li, Q. Li, Y. Ye, S. Xu, Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics'2019 challenge. *CoRR*. **abs/1812.08108** (2018). <http://arxiv.org/abs/1812.08108>
- D. Li, R. Baral, T. Li, H. Wang, Q. Li, S. Xu, Hashtran-dnn: a framework for enhancing robustness of deep neural networks against adversarial malware samples. *CoRR*. **abs/1809.06498** (2018). <http://arxiv.org/abs/1809.06498>
- Z. Li, D. Zou, S. Xu, X. Ou, H. Jin, S. Wang, Z. Deng, Y. Zhong, in *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. Vuldeepecker: A deep learning-based system for vulnerability detection (Internet Society, San Diego, 2018)
- Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, Z. Chen, S. Wang, J. Wang, Sysevr: A framework for using deep learning to detect software vulnerabilities. *CoRR*. **abs/1807.06756** (2018). <http://arxiv.org/abs/1807.06756>
- G. Grieco, G. L. Grinblat, L. Uzal, S. Rawat, J. Feist, L. Mounier, in *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy. CODASPY '16. Toward large-scale vulnerability discovery using machine learning* (ACM, New York, 2016), pp. 85–96
- Z. Li, D. Zou, S. Xu, H. Jin, H. Qi, J. Hu, in *Proceedings of the 32nd Annual Conference on Computer Security Applications, ACSAC 2016, Los Angeles, CA, USA, December 5-9, 2016*. Vulpecker: an automated vulnerability detection system based on code similarity analysis (ACM, Los Angeles, 2016), pp. 201–213
- Y. Chen, M. Khandaker, Z. Wang, in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ASIA CCS '17. Pinpointing vulnerabilities* (ACM, New York, 2017), pp. 334–345
- J. D. Cryer, K.-S. Chan, *Time Series Analysis With Applications in R*. (Springer, New York, 2008)
- P. J. Brockwell, R. A. Davis, *Introduction to Time Series and Forecasting*. (Springer, Switzerland, 2016)
- J. Ke, H. Zheng, H. Yang, X. M. Chen, Short-term forecasting of passenger demand under on-demand ride services: A spatio-temporal deep learning approach. *Transp. Res. C Emerg. Technol.* **85**, 591–608 (2017)
- M. Barabas, G. Boanea, A. B. Rus, V. Dobrota, J. Domingo-Pascual, in *Intelligent Computer Communication and Processing (ICCP), 2011 IEEE*

- International Conference On. Evaluation of network traffic prediction based on neural networks with multi-task learning and multiresolution decomposition* (IEEE, Cluj-Napoca, 2011), pp. 95–102
33. A. Azzouni, G. Pujolle, A Long Short-Term Memory Recurrent Neural Network Framework for Network Traffic Matrix Prediction. CoRR. **abs/1705.05690** (2017). <http://arxiv.org/abs/1705.05690>
 34. S. Siami-Namini, A. S. Namin, Forecasting Economics and Financial Time Series: ARIMA vs. LSTM. CoRR. **abs/1803.06386** (2018). <http://arxiv.org/abs/1803.06386>
 35. C.-M. Kuan, T. Liu, Forecasting exchange rates using feedforward and recurrent neural networks. *J. Appl. Econ.* **10**(4), 347–364 (1995)
 36. T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, S. Khudanpur, in *Proceedings of the 11th Annual Conference of the International Speech Communication Association*. Recurrent neural network based language model (International Speech Communication Association (ISCA), Makuhari, Chiba, 2010), pp. 1045–1048
 37. M. Sundermeyer, I. Oparin, J. L. Gauvain, B. Freiberger, R. Schlüter, H. Ney, in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. Comparison of feedforward and recurrent neural network language models (IEEE, Vancouver, 2013), pp. 8430–8434
 38. Z. Huang, G. Zweig, B. Dumoulin, in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Cache based recurrent neural network language model inference for first pass speech recognition (IEEE, Florence, 2014), pp. 6354–6358
 39. X. Liu, Y. Wang, X. Chen, M. J. Gales, P. C. Woodland, in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference On*. Efficient lattice rescoring using recurrent neural network language models (IEEE, Florence, 2014), pp. 4908–4912
 40. M. Schuster, K. K. Paliwal, Bidirectional recurrent neural networks. *IEEE Trans. Sig. Process.* **45**(11), 2673–2681 (1997)
 41. Y. Bengio, P. Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Netw.* **5**(2), 157–166 (1994)
 42. S. Hochreiter, J. Schmidhuber, Long short-term memory. *Neural Comput.* **9**(8), 1735–1780 (1997)
 43. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. (MIT Press, MA, 2016)
 44. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization. CoRR. **arXiv preprint arXiv:1412.6980** (2014)
 45. R. J. Hyndman, A. B. Koehler, Another look at measures of forecast accuracy. *Int. J. Forecast.* **22**(4), 679–688 (2006)
 46. P. Baecher, M. Koetter, T. Holz, M. Dornseif, F. Freiling, in *International Workshop on Recent Advances in Intrusion Detection*. The nepenthes platform: An efficient approach to collect malware (Springer, Berlin, Heidelberg, 2006), pp. 165–184
 47. S. Almotairi, A. Clark, G. Mohay, J. Zimmermann, in *2008 IFIP International Conference on Network and Parallel Computing*. Characterization of attackers' activities in honeypot traffic using principal component analysis (IEEE, Shanghai, 2008), pp. 147–154
 48. G. P. Zhang, Time series forecasting using a hybrid arima and neural network model. *Neurocomputing.* **50**, 159–175 (2003)
 49. M. Kumar, M. Thenmozhi, Forecasting stock index returns using arima-svm, arima-ann, and arima-random forest hybrid models. *Int. J. Bank. Account. Financ.* **5**(3), 284–308 (2014)
 50. J. Friedman, T. Hastie, R. Tibshirani, *The Elements of Statistical Learning, vol. 1*. (Springer, New York, 2001)
 51. P.-F. Pai, C.-S. Lin, A hybrid arima and support vector machines model in stock price forecasting. *Omega.* **33**(6), 497–505 (2005)
 52. Y. Chen, B. Yang, J. Dong, A. Abraham, Time-series forecasting using flexible neural tree model. *Inf. Sci.* **174**(3-4), 219–235 (2005)

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com