A comparative study of the most important methods for forecasting the ICT systems vulnerabilities

--Technical report--

O. Cosma, M. Macelaru, P.C. Pop, C. Sabo and I. Zelina

Technical University of Cluj-Napoca, North University Center of Baia Mare, Dr. V. Babes 62A, 430083, Romania

Abstract. Nowadays, companies are facing plenty of IT secure attacks and to guarantee safe, untroubled, and continuous functioning of their business, they should detect and forecast the volume of IT security vulnerabilities and be prepared for future threats. The aim of this paper is to present a comparative study of the most important and promising methods for forecasting the ICT systems vulnerabilities.

Keywords: Security vulnerabilities, Forecasting, Time series forecasting, Neural networks.

1 Introduction

Due to the exponential growing trend in ICT system vulnerabilities and cyber threats, security became more and more important. Therefore, the companies have become preoccupated with the prediction, forecasting and propagation of the vulnerabilities.

A computer system vulnerability can be defined as a weakness within the system or network that might be taken advantage of, to generate damage or to permit attackers to exploit the network in some way. The vulnerabilities may appear because of unexpected intercommunications between distinct software programs, network components or weakness of an individual program. Vulnerabilities exist in every network, and it is impossible to find and to focus on all of them due to the highly complex structure of modern network architecture.

Even though there are different approaches to cover the process of software vulnerability analysis and discovery, most of them are approximate solutions with lack of soundness or completeness, or even both [27]. In this sense the previous research lines focus on providing an improvement on some specific aspects of the process, as the precision, efficiency, or vulnerability coverage.

In this paper we will focus on the forecasting the ITC systems vulnerabilities. The existing methods from the literature for forecasting ITC systems vulnerabilities can be classified in three categories:

1. Time series analysis-based models: Autoregressive integrating moving average (ARIMA), Exponential Smoothing, etc.

2. Artificial intelligence-based models: Neural Networks, Support Vector Machine (SVM), Bayesian Network, k-Nearest Neighbor, etc.
3. Statistical based models: Regression techniques, Linear Regression, Logistic Regression, Random Forest, Naïve Bayes, Decision Tree, Least Mean Square, Reliability Growth models, Statistical Code analysis, etc.

For a comprehensive survey on forecasting ITC systems vulnerabilities, we refer to Yasasin et al. [6], Roumani et al. [12].

The aim of our paper is to present a comparative study of the most important and promising methods for forecasting ITC systems vulnerabilities. We describe six methods belonging to the first two categories of models.

The remaining of our paper is organized as follows: in Section 2 we describe in detail the considered methods for forecasting ITC systems vulnerabilities and in Section 3 we present our conclusions and as well some future research directions.

2 Methods of forecasting the ICT systems vulnerabilities

2.1 Time series forecasting based models

A *time series* is a series of data points indexed in time order and are usually plotted using run charts. Time series have several applications such as: economic forecasting, stock market analysis, inventory studies, weather forecasting, earthquake predictions, etc. *Time series analysis* contains techniques for examining time series data in order to obtain relevant statistics and other features from the data. *Time series forecasting* makes use of a model to forecast future values based on previously noticed values. For more information on this topic, we refer to Hyndman and Athanosoupulos [13].

Autoregressive integrating moving average (ARIMA) models

ARIMA models are advanced statistical models and are considered to be the most general models for forecasting time series that can be made stationary. ARIMA models give descriptions of the autocorrelations in the data [13] and not description of trend and seasonality as exponential smoothing methodologies do.

A stationary time series is a time series whose statistical properties do not rely on the time at which the series is noticed, they remain constant over time. Time series with trends or seasonality are not stationary. If nonstationary data is used, the estimators don't have the asymptotic normality and consistency properties, for the time series models. When an ARIMA model is constructed, the first step is to decide if the time series is stationary. If the series to be studied is not stationary it is transformed in order to become stationary.

The ARIMA models include autoregressive terms (AR), moving average terms (MA) and differencing operations (Integrated). Autoregression means regression of the variable against itself. In an autoregression model, the value of the variable is forecast using a linear combination of its past values. An autoregressive model of order p, AR(p), for a time series y, where y_t is the value of the variable at moment t, is described

as $y_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \cdots + \varphi_p y_{t-p} + \varepsilon_t$, where ε_t is white noise. The parameters $\varphi_1, \varphi_2, \ldots, \varphi_p$ are the autocorrelation coefficients and they define the time series patterns. For *p=1* the constraint for parameter φ_1 is $-1 < \varphi_1 < 1$ and for *p=2* the constraints for the parameters are $1 < \varphi_2 < 1$, $\varphi_1 + \varphi_2 < 1, \varphi_2 - \varphi_1 < 1$.

The moving average model MA of order *q*, MA(q), uses past forecast errors instead of using previously observed values of the forecast variable, $y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}$.The parameters $\theta_1, \theta_2, \ldots, \theta_q$ define the time series patterns. The constraints for these parameters are $-1 < \theta_1 < 1$ for *p*=1 and $-1 < \theta_2 < 1$, $\theta_1 + \theta_2 > -1$, $\theta_1 - \theta_2 < 1$ for *p*=2.

If the time series is non-stationary, differencing is a way to make it stationary. Differencing means computing the differences between consecutive observations in the non-stationary series and the differenced series terms $y'_t = y_t - y_{t-1}$ describe the change between two consecutive terms in the original series.

If the differenced data is not stationary, then we can repeat the differentiation. The data can be differenced again to obtain a stationary series: $y''_t = y'_t - y'_{t-1} = y_t - 2t_{t-1} + y_{t-2}$. The process of differentiation can be repeated until the series become stationary. Usually, first or second differentiation is enough to build good models.

A general ARIMA (*p, d, q*) model is obtained combining differencing with autoregression and moving average model, and *p* is the number of autoregressive terms, *d* is the number of differences necessary for stationarity and *q* is the number of lagged forecast errors [13]: $y'_t = c + \varphi_1 y'_{t-1} + \cdots + \varphi_p y'_{t-p} + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t$.

Exponential Smoothing

Exponential smoothing is a popular forecasting method that aims to produce a smooth Time Series. The method was proposed by Brown [14], Holt [15] and Winter [16]. Its main feature is that it assigns higher weight in forecasting to recent observations than to older observations. Unlike the ARIMA models, exponential smoothing model do not require the time series to be stationary.

Single exponential smoothing (SES) is the simplest of the exponential smoothing methods and is applied to data series with no trend or seasonality. The equation in this case is $y_t = \alpha y_{t-1} + (1-\alpha) y_t - 2$, where $0 \leq \alpha \leq 1$ is the smoothing parameter.

Double exponential smoothing (DES) proposed by Holt [15] is an extension of SES and is used to forecast data series with trend. The smoothing equations in this case are $L_t = \alpha y_t + (1-\alpha)(L_{t-1} + b_{t-1})$, $b_t = \gamma(L_t - L_{t-1}) + (1-\gamma) b_{t-1}$, where α, γ are the smoothing parameters for the level and for the trend, $0 \leq \alpha, \gamma \leq 1$, L_t is the level of the series at time *t* and b_t is the trend of the series at time *t.* The forecast equation is $y_{t+1} = L_t + b_t$, meaning that the 1 step ahead forecast is the sum between the estimated level at time *t* and the trend value at time *t.*

To also capture the seasonality, Holt and Winters [1] proposed a model with smoothing equations for level, for trend and for seasonality. We present next the Holt-Winters additive exponential smoothing model that was proposed by Roumani et al. [12] for vulnerability analysis and forecasting. The equations that describe the Holt-Winters additive exponential smoothing model are:

$$L_t = \alpha(Y_t - S_{t-s}) + (1 - \alpha)(L_{t-1} - b_{t-1}) \quad (1)$$

$$b_t = \gamma(L_t - L_{t-1}) + (1 - \gamma)b_{t-1} \quad (2)$$

$$S_t = \delta(Y_t - L_t) + (1 - \delta)\, S_{t-s} \quad (3)$$

$$F_{t+m} = L_t + mb_t + S_{t+m-s} \quad (4)$$

where α, γ and δ are the smoothing parameters, Y_t represents the number of vulnerabilities at time t calculated in months, m represents the number of future periods to predict (12 months), s represents the length of the seasonality (12 months), L_t represents the level of the series at time t, b_t represents the tendency of the series at time t and S_t is the seasonal component at the time t.

The first equation (level) describes the relative magnitude of the number of vulnerabilities, the second one (trend) describes the gradual upward or downward long-term movement of the number of vulnerabilities, the third one (seasonality) describes the short-term regular variations of the number of vulnerabilities at regular intervals and finally the last equation describes the vulnerability for a given period m.

2.2 Artificial intelligence-based models

The foundations of artificial neural networks (ANNs) were established in 1943 by Warren McCulloch and Walter Pitts. They described in [1] how neurons might work. The back propagation algorithm (BP) for multilayer ANNs was proposed by Rummelhart et al. in 1986 [2].

ANNs are inspired by the operation of the human brain. They are made up of a set of interconnected components in several layers, called neurons or perceptrons. The inputs of each neuron are connected to the outputs of neurons in the previous layer. The neurons in the first layer get the inputs of the model, and the outputs of the neurons in the final layer provide the result. The structure of a neuron is presented in Fig. 1.a, where $x_i, i = 1, n$ are the inputs, $w_i, i = 1, n$ are the weights associated to the inputs, b is the bias, f is the activation function and y is the output. The neuron operation is defined by relation (5). It calculates a weighted sum of the inputs and bias, and then it passes it through the activation function to determine the output. The most widely used activation functions are *Identity* (6), *Sigmoid* (7), *ReLu* (8), *Tanh* (9) and *Softplus* (10).

$$y = f(b + \sum_{i=1}^{n} w_i x_i) \quad (5)$$

$$f_1(x) = x \quad (6)$$

$$f_2(x) = \frac{1}{1+e^{-x}} \quad (7)$$

$$f_3(x) = \max(0, x) \quad (8)$$

$$f_4(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (9)$$

$$f_5(x) = \ln(1 + e^x) \quad (10)$$

Multilayer Perceptron Models

A Multilayer Perceptron (MLP) model with an input layer, a hidden layer, and an output layer is shown in Fig. 1.b. The outputs of the model are given by relations (11) and (12), where f is the activation function of the output layer neurons and g is the activation function of the hidden layer neurons. A more compact representation of this model is given in Fig. 2.a, where x and y represent the input and the output vectors, W and V are the hidden and the output layers weights matrices, b^h and b^o are the hidden and the output layers bias vectors, h is the hidden layer output vector, and g and f are the activation functions of the hidden and the output layers. The output is given by relations (13) and (14), that are compact representations of (11) and (12). The model can be extended by adding more hidden layers. Training such a model involves adjusting the weights and biases in such a way as to provide an output as close as possible to the correct value for each input. The most widely used methods of training ANNs are based on the Gradient Descent – Back Propagation (GD-BP) algorithms [3] or genetic algorithms.

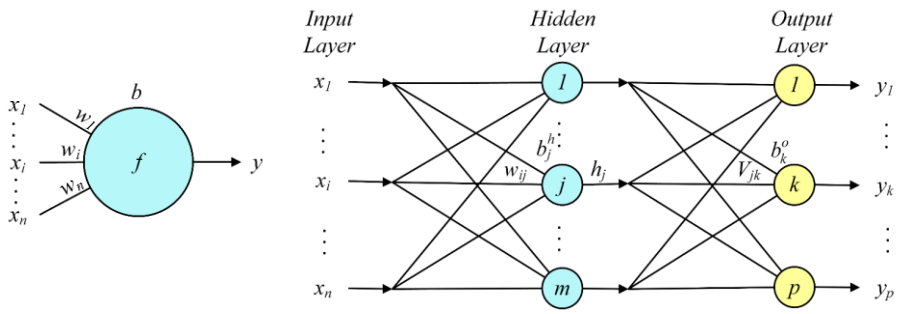


Fig. 1. a.) Perceptron structure, b.) MLP model with one hidden layer.

$$\mathrm{h}_j = g\left(b_j^h + \sum_{i=1}^{n} W_{ij} x_i\right), j = 1, \dots, m \qquad (11)$$

$$h = g(b^h + xW) \qquad (13)$$

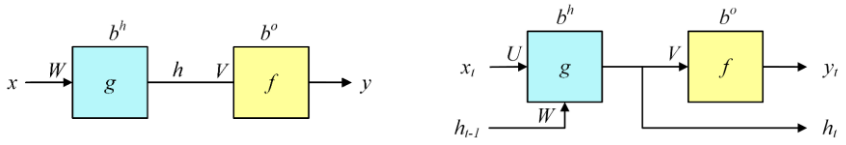
$$\mathrm{y}_k = f\left(b_k^o + \sum_{j=1}^{m} V_{jk} h_j\right), k = 1, \dots p \qquad (12)$$

$$y = f(b^o + hV) \qquad (14)$$

Recurrent Neural Networks

Recurrent Neural Nets RNNs were designed to process sequential data. They can memorize previous states and use them to determine the current state. The structure of a RNN is shown in Fig. 2.b. Its state is composed by the hidden layer output vector h. The state vector h calculated at step $t-1$ is processed as an entry at step t. The total number of trainable parameters does not depend on the number of steps, but only on the dimensions of the layers. Thus, for the network structure in Fig. 2.b, the total number of parameters is $(n+m+1)m+(m+1)p$, where n is the size of the input vector x, m is the size of the status vector h, and p is the size of the output vector y. The

outputs at step t are given by relations (15) and (16). They depend not only on the entries at step t, but on their entire evolution, starting from the initial step.



a.) b.)

Fig. 2. a.) Compact representation of the MLP model with one hidden layer, b.) RNN structure.

$$h_t = g(b^h + x_t\mathrm{U} + h_{t-1}\mathrm{W}) \quad (15) \qquad y_t = f(b^o + h_t\mathrm{V}) \quad (16)$$

Long Short-Term Memory Models

The main disadvantage of RNNs lies in the fact that they cannot learn long sequences, because of the vanishing gradient problem [3]. This problem occurs in deep networks with many hidden layers. RNNs do not have long-term memory. The Long Short-Term Memory (LSTM) models have been projected to solve this problem. The structure of a LSTM cell is presented in Fig. 3, where f is the forget gate, i is the input gate, g is the input updater, O is the output gate, C_t is the current cell state and h is the hidden state. Its operation is defined by relations (17) – (22).

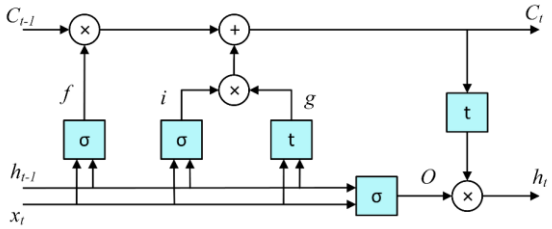


Fig. 3. LSTM cell structure.

$$f = \sigma(x_t U^f + h_{t-1} W^f) \quad (17)$$

$$i = \sigma(x_t U^i + h_{t-1} W^i) \quad (18)$$

$$g = \tanh(x_t U^g + h_{t-1} W^g) \quad (19)$$

$$C_t = C_{t-1} \circ f + g \circ i \quad (20)$$

$$O = \sigma(x_t U^o + h_{t-1} W^o) \quad (21)$$

$$h_t = \tanh(C_t) \circ O \quad (22)$$

Convolutional Neural Networks

The MLP models contain several intermediate layers that are fully connected with the adjacent layers. Convolutional Neural Networks (CNN) are different in this respect. After the input layer follow several layers of different types [3]. The first type of layer is the convolution layer, that applies convolution operations, using a set of small filters, that contain weights trained to capture different features. The output of each filter is an activation map. The output of a convolution layer is composed of all the activation maps created by its filters. A convolution layer is sparsely connected with the previous layer.

Another type of layer in a CNN is the pooling layer, also sparsely connected with the previous layer. It has the purpose to downsample data, based on different strategies, such as: max poling designed to catch the peaks, or average pooling.

The third type of layer in a CNN is the dense layer, that is fully connected with the previous layers. This type of layer is similar with the hidden layers of the MLP model. The dense layers are usually placed at the end of the CNN model, before the output layer.

The common way of training a CNN is the GD-BP algorithm.

2.3 ICT vulnerabilities forecasting

Roumani et al. [12] used time series analysis to build predictive models for five well-known browsers: Chrome, Firefox, Internet Explorer, Safari and Opera, on a collection of vulnerability datasets from National Vulnerability Database (NVD) and concluded that ARIMA is the best fit vulnerability model.

Yasasin et al. [6] used different methodologies to predict IT vulnerabilities of different system and software packages (operating systems, browsers, and office solutions). The conclusion is that the ARIMA method achieved low forecasting errors for all types of software that have been investigated and therefore it is recommended for forecasting software vulnerabilities.

An ANN-based time series forecasting model uses observations at previous times as inputs, and the outputs of the model represent the forecast values. Neural networks have multiple advantages over classic models. They can produce quality forecasts, even if they work with the original data. There is no need to eliminate trend and seasonality. The model can be easily generalized to perform multivariate forecasts, by increasing the number of inputs. ANN can implicitly detect complex nonlinear relationships between dependent and independent variables.

The main disadvantage of ANN lies in the fact that their training is a complex process that requires a lot of processing power, and they can be overfitted.

Feed Forward-ANNs have been successfully used both to forecast software systems vulnerabilities and in other forecasting applications. The experimental data needed to train the vulnerability forecasting models are taken from the National Vulnerability Database (NVD) [11] or other public sources.

Most of the proposed models contain a single hidden layer of neurons [4, 5, 6, 7], but there are also variants with several intermediate layers [8]. There are several proposals that present optimization strategies to find the best dimension of the input and intermediary layers [4, 5].

The majority of the proposed models are trained by the Gradient Descent and the Back Propagation (GD-BP) algorithms [5, 7, 9], but there are also some proposals of evolutionary computation training [8, 9] and hybrid training algorithms that use an adaptive differential evolution algorithm [10] or a particle swarm optimization algorithm [9] in combination with GD-BP. These proposals speculate that evolutionary computational techniques are making significant progress in the first part of the optimization process and manage better to avoid local minima than GD-BP. When evolution begins to stagnate, they switch to GD-BP, which reaches the optimal solution faster.

Most of the proposed models use the sigmoid activation function [6, 7] or the tanh activation function [5] in the hidden layer, but there are other proposals, such as the sinusoidal function [8]. Almost all proposed models use the identity activation function in the output layer, but there are also solutions based on the modified sigmoid activation function [7].

The proposed forecasting models are usually compared with other known techniques (Exponential smoothing, Croston's methodology, ARIMA, Support Vector Machines, Vulnerability Discovery Models) in terms of forecast accuracy and forecast bias. Most of the models trained by GD-BP are outperformed by the models they are compared with [4, 6], but there are also exceptions [5]. The models trained with evolutionary computation or hybrid algorithms outperform the other models [8, 9, 10].

The best forecast models proposed recently have LSTM cells in their composition and are trained by BP-GD. A comparative analysis of several types of deep neural networks is presented in [23], and it is concluded that the Convolutional Neural Networks (CNN), MLP, RNN and LSTM models perform well for one step forecasting and less satisfactory for multiple steps forecasting. Another comparative analysis of different forecasting models is presented in [25]. The bottom line is that LSTM models give the best forecasts, but CNN models are the most robust to changes in configuration parameters. Various forecast models are compared with LSTM models in [18, 21 and 22]. In general LSTM offers the best performance, except for the [18] work, in which CNN and shallow ANN models are found to be better. Different hybrid CNN – LSTM models are proposed in [17, 19, 20, 24, 26]. They outperform the other models they are compared with (MLP, CNN, RNN, LSTM, ARIMA) in all works, even though some authors report longer training times.

3 Conclusions

In this article we have presented some of the forecasting techniques that have been proven to be effective in the case of software vulnerabilities. In order to have an overview of the latest results in the field, we have not strictly limited our study to software vulnerabilities, but we have also considered other applications for forecasting natural phenomena. It is difficult to draw a conclusion because the analyzed works used different data sets. An objective conclusion could be drawn only if all models were tested under the same conditions, on the same data sets. Most authors have implemented several different models to compare their effectiveness, or to compare their proposed

methods with the existing ones. However, it is known that the performance of a model depends on the parameters of the model, how it is optimized, how the experimental data is preprocessed, how the training is done, etc.

Overcoming all these uncertainties, based on the results analyzed we can conclude that the latest CNN – LTSM hybrid forecasting models seem to be the most accurate, but they require more complex training. However, the performance of this models must also be proven in the case of software vulnerabilities forecasting. There are several studies that recommend the classic Croston and ARIMA models, as they consistently obtain quality predictions. Several studies show that superior performance can be achieved by changing the GD-BP with other algorithms, such as evolutionary, hybrid, or other types, for training the ANN models. Another conclusion is that CNN, MLP, RNN and LSTM models perform well for one step forecasting and less satisfactory for multiple steps forecasting.

Starting from this study, we aim to test the efficiency of hybrid ANN models for forecasting software vulnerabilities, and to use other training techniques, different from GD – BP, that have proven their effectiveness for the FF-ANN models.

Acknowledgment. This work was supported by the project BIECO (www.bieco.org) that received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 952702.

References

1. Warren S. McCulloch, Walter Pitts: A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5, 115–133 (1943)
2. Rumelhart, D., Hinton, G. & Williams, R.: Learning representations by back-propagating errors. Nature 323, 533–536 (1986).
3. Venkata Reddy Konasani, Shailendra Kadre: Machine Learning and Deep Learning Using Python and Tensor Flow. McGraw Hill: New York (2021)
4. Pokhrel, N.R., Rodrigo, H., Tsokos, C.P.: Cybersecurity:Time Series Predictive Modeling of Vulnerabilities of Desktop Operating System Using Linear and Non-Linear Approach. Journal of Information Security, 8, 362-382 (2017).
5. Movahedi, Y., Cukier, M., Gashi, I.: Vulnerability prediction capability: A comparison between vulnerability discovery models and neural network models, Computers & Security 87, 101596 (2019).
6. Emrah Yasasin, Julian Prester, Gerit Wagner, Guido Schryen: Forecasting IT security vulnerabilities –An empirical analysis, Computers & Security 88, 101610 (2020).
7. Christopher Bennett, Rodney A. Stewart, Cara D. Beal: ANN-based residential water end-use demand forecasting model. Expert Systems with Applications 40, 1014–1023 (2013).
8. Pei-Chann Chang, Di-di Wang, Chang-le Zhou: A novel model by evolving partially connected neural network for stock price trend forecasting. 39, 611–620 (2012).
9. Jing-Ru Zhang, Jun Zhang, Tat-Ming Lok, Michael R. Lyu: A hybrid particle swarm optimization–back-propagation algorithm for feedforward neural network training. Applied Mathematics and Computation 185 1026–1037 (2007).

10. Lin Wang, Yi Zeng, Tao Chen: Back propagation neural network with adaptive differential evolution algorithm for time series forecasting. Expert Systems with Applications 42, 855–863 (2015).
11. National Vulnerability Database, https://nvd.nist.gov/, last accessed 2021/04/24.
12. Roumani Y., Nwankpa J.K., Roumani Y.F.: Time series modeling of vulnerabilities, Computers & Security 51, 32-40 (2015).
13. Hyndman R., Athanosoupulos G.: Forecasting: Principles & Practice, 2nd edition, OTexts: Melbourne, Australia. OTexts.com/fpp2 (2021).
14. Brown, R. G.: Statistical forecasting for inventory control, McGraw/Hill, (1959).
15. Holt, C. E.: Forecasting seasonals and trends by exponentially weighted averages (O.N.R. Memorandum No. 52). Carnegie Institute of Technology, (1957).
16. Winters, P. R.: Forecasting sales by exponentially weighted moving averages. *Management Science*, *6*(3), 324–342, (1960).
17. Wenjie Lu, Jiazheng Li, Yifan Li, Aijun Sun, Jingyang Wang: A CNN-LSTM-Based Model to Forecast Stock Prices, *Complexity*, 6622927 (2020).
18. AndreasWunsch, Tanja Liesch, Stefan Broda: Groundwater level forecasting with artificial neural networks: a comparison of long short-term memory (LSTM), convolutional neural networks (CNNs), and non-linear autoregressive networks with exogenous input (NARX), Hydrology and Earth System Sciences, 25(3), 1671–1687 (2021).
19. Rob Shipman, Rebecca Roberts, Julie Waldron, Sophie Naylor, James Pinchin, Lucelia Rodrigues, Mark Gillott: We got the power: Predicting available capacity for vehicle-to-grid services using a deep recurrent neural network, Energy 221, 119813 (2021).
20. Meng Ma, Zhu Mao: Deep-Convolution-Based LSTM Network for Remaining Useful Life Prediction, IEEE Transactions on Industrial Informatics, 17(3) 1658-1667, (2021).
21. Eric Hitimana, Gaurav Bajpai, Richard Musabe, Louis Sibomana, Jayavel Kayalvizhi: Implementation of IoT Framework with Data Analysis Using Deep Learning Methods for Occupancy Prediction in a Building, Future Internet, 13, 67 (2021).
22. Widodo Budiharto: Data science approach to stock prices forecasting in Indonesia during Covid-19 using Long Short-Term Memory (LSTM), Journal of Big Data 8, 47, (2021).
23. Rohit Kaushik, Shikhar Jain, Siddhant Jain, Tirtharaj Dash: Performance evaluation of deep neural networks for forecasting time-series with multiple structural breaks and high volatility, CAAI Transactions on Intelligence Technology, 1–16 (2021).
24. Somu, N., Raman, G.M.R., Ramamritham, K: A deep learning framework for building energy consumption forecast, Renewable and Sustainable Energy Reviews, 137, 110591 (2021).
25. Pedro Lara-Benitezy, Manuel Carranza-Garcia, Jose C. Riquelme: An Experimental Review on Deep Learning Architectures for Time Series Forecasting, International Journal of Neural Systems, 31(3), 2130001 (2021).
26. Ioannis E. Livieris, Niki Kiriakidou, Stavros Stavroyiannis, Panagiotis Pintelas: An Advanced CNN-LSTM Model for Cryptocurrency Forecasting, Electronics 10, 287 (2021).
27. Ranjit Jhala, Rupak Majumdar: Software model checking. ACM Computing Surveys, article no. 21, (2009).