IJRRS
International Journal of Reliability, Risk Safety: Theory and Application

Online ISSN: 2676-3346

Vol. 4/ Issue 2/ 2021/ pp. 1-12
DOI : 10.30699/IJRRS.4.2.1
Received: 2021.09.21, Accepted: 2022.03.07

Original Research Article

LSTM Encoder-Decoder Dropout Model in Software Reliability Prediction

Shahrzad Oveisi[1], Ali Moeini[1*], Sayeh Mirzaei[1]

1. Department of Algorithms and Computation, School of Engineering Sciences, College of Engineering, University of Tehran, Tehran, IRAN

[*]**moeini@ut.ac.ir**

Abstract

Numerous methods have been introduced to predict the reliability of software. In general, these methods can be divided into two main categories, namely parametric (e.g. software reliability growth models) and non-parametric (e.g. neural networks). Both approaches have been successfully implemented in software testing applications over the past four decades. Since most software reliability prediction data are available in the form of time series, deep recurrent network models (e.g. RNN, LSTM, NARX, and LSTM Encoder-Decoder networks) are considered as powerful tools to be employed in reliability-related problems. However, the problem of overfitting is a major concern when using deep neural networks for software reliability applications. To address this issue, we propose the use of dropout; therefore, this study utilizes a deep learning model based on LSTM Encoder-Decoder Dropout to predict the number of faults in software and assess software reliability. Experimental results show that the proposed model has better prediction performance compared with other RNN-based models.

Keywords: LSTM; LSTM Encoder-Decoder; NARX, RNN; Dropout; Software Reliability Prediction; Bayesian.

1. Introduction

Today, the software has found important roles in systems with sensitive applications such as medicine, the military industry, and nuclear reactors. Therefore, the risks and effects of software failures can lead to irreversible damage [1-4]. The existence of such issues has driven software system developers to check the quality of software before launching the product to the market as well as during the project completion stages [5-6].

In general, there are two groups of reliability growth models: *parametric* and *non-parametric*. A majority of parametric models are based on the non-homogenous Poisson process (NHPP), which has been extensively used in software reliability engineering. While NHPP models are widely used, they are always associated with certain limitations. Such limitations usually involve a series of basic assumptions regarding the random nature of software fault, which may not be completely observed in real-life conditions. Several alternative solutions have been introduced to overcome this problem, which falls into the group of non-parametric models. In non-parametric models like neural networks (NN), statistical failure data are a basis for prediction [7].

For instance, neural networks are capable of extracting a model from the existing datasets of fault processes. There are several instances of successful use of neural networks to solve software reliability problems. In all these approaches, the main concern is to augment the accuracy of the model for fault prediction so that the software delivery process is attained with more confidence.

SRGM models are described according to several parameters, and the selection of appropriate estimation of these parameters plays an important role in the eventual accuracy of the model. The use of the traditional neural network (NN) [8] based models for software reliability prediction [9] is not efficient enough. Overfitting is a major disadvantage of neural networks. In addition, it is difficult to adjust their weight parameters which requires several learning tricks. For this reason, the predictive performance of these models is always affected by the mentioned problems. In recent years, deep neural networks and their diverse architectures have been extensively studied and used to overcome the weaknesses of simple neural networks. In addition to taking advantage of the capabilities of traditional neural networks, deep neural networks have shown much higher performance in the field of prediction in practice [10]. In particular, a group of neural network methods has been widely used in time

series prediction issues. Convolutional neural networks (CNN) [11], recurrent neural networks (RNN) [12], long short-term memory networks (LSTM) [13-16], nonlinear autoregressive with external input (NARX) [17] are among these networks.

Recurrent neural networks are a special class of deep neural networks that are characterized by internal recurrent communication, have the ability to model nonlinear dynamic systems, and have been used in various predictive actions. More precisely, they perform better than conventional time-series predictor models. A recurrent neural network processes sequential information and performs the same operation on any part of the input sequence. Its input, at each time stage, depends on previous inputs and previous calculations; hence, networks integrate past and present information and predict future values. This feature gives the network the ability to expand the memory on previous data that is different from the neural network, which assumes that all its inputs are independent of each other. Theoretically, recurrent neural networks can remember long sequences conventionally and optionally.

One should also pay attention to the available data set when choosing the appropriate method. In this paper, according to the data used, LSTM, LSTM Encoder-Decoder based model, NARX, and RNN networks are used for software reliability prediction.

Deep neural networks consist of several hidden nonlinear layers, which have caused these networks to become highly expressive models that can learn the complex relationships between their inputs and outputs. If the available training data is limited, these complex relationships can lead to sampling noise, so complicated relationships may exist in the training set but not in the real data set even when the distribution of both datasets is the same. This causes overfitting, and many methods have been introduced to reduce the overfitting problem such as the dropout method, which is discussed in this paper.

Dropout is a regularization method where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while training a network [16,18]. This has the effect of reducing overfitting and improving model performance [18].

To evaluate the proposed method, we applied it to 3 datasets, including the Launch Abort System (LAS), namely a space vehicle LAS with functional dependencies among the elements. This system consists of two main subsystems: The emergency Detection System (EDS) as a module of the air/space vehicle system and the Launch Escape System (LES). EDS is used for detecting abnormal and emergency conditions in a flying vehicle and is designed to treat mechanical and electrical failures in each of the propulsion, electrical, and control systems.

In this paper, we consider the fault detection time as an input time series and the cumulative number of detected faults as an output sequence. Furthermore, we use LSTM Encoder-Decoder-based model to capture the features from fault datasets and to predict the next-step and end-point fault number. The experimental results indicate that the proposed model can be used to accurately predict the number of faults in software and evaluate software reliability. In addition, software reliability will be higher when the detected faults are increased. In other words, when more faults are detected, the probability of software failures becomes lower. Therefore, accurately predicting the number of software faults is of importance for ensuring the reliability of software.

The rest of this paper is organized as follows. After the introduction, the research background is presented in Section 2. Section 3 introduces research fundamentals. Next, the main contributions of the paper are explained in Section 4. A detailed case study using Launch Abort System (LAS) and other datasets is provided in Section 6 together with the quantitative analysis based on the proposed method. Finally, we conclude the paper in Section 7.

2. Research Background

So far, several types of research have been done regarding prediction using neural networks and recurrent neural networks as well as the use of dropout in neural networks. In this section, we briefly survey some of these studies.

A surround vehicle motion prediction algorithm has been presented in [19] for multi-lane turn intersections using a Long Short-Term Memory (LSTM)-based Recurrent Neural Network (RNN).

A novel model called LC-RNN has been presented in [20] to achieve more accurate traffic speed prediction than existing solutions. It takes advantage of both RNN and CNN models by a rational integration of them to learn more meaningful time-series patterns that can adapt to the traffic dynamics of surrounding areas.

Srivastava [21] reviewed dropout with feed-forward neural nets, as well as a dropout with Boltzmann machines and marginalizing dropout. Sutskever et al. [22] used deep neural networks to improve the overall sequential learning problem and proposed an end-to-end learning method (sequence mapping) for machine translation. Pascanu et al. [23] have studied several deep recurrent neural networks and proposed a new framework for these networks. Wang et al. [24] used the Deep Belief Neural Network model to record the semantic characteristics of the program's abstract syntax tree (ASTs) and used this model to predict or detect software faults. Bhuyan et al. [25] proposed a detailed feed-forward back propagation network model for predicting reliability using failure data, and the obtained results showed a good fit compared with other models. Jinyong Wang et al. [26] suggested a deep learning model based on the recurrent neural network (RNN) Encoder-Decoder for software reliability prediction, the architecture of which uses one

input layer, two hidden layers, and one output layer. As shown in the paper, the obtained results are optimal compared with four neural networks (NN) models using 14 fault datasets in terms of end-point predictions and next-step predictions.

Recent advances in CNNs for time series prediction include [27] where the authors propose an undecimated convolutional network for time series modeling based on the undecimated wavelet transform and [28] in which the authors suggest the use of an autoregressive-type weighting system for forecasting financial time series where the weights are allowed to be data-dependent by learning them through a CNN. In general, literature on financial time series forecasting with convolutional architectures is still scarce as these types of networks are much more commonly applied in classification problems.

3. Research Fundamentals

In this section of the paper, we briefly review a number of required backgrounds before starting the main parts of the paper.

3.1 Dropout

Dropout prevents overfitting and provides a way to effectively combine a large number of different neural network architectures with an almost exponential order. This method refers to the dilution of units in the neural network. Diluting a unit is meant to temporarily remove that unit from the network along with all input and output connections. The units to be diluted are randomly determined. In the simplest case, each unit with a constant probability (p) is kept independent of the other units, in which p can be selected using a validation set and easily adjusted to 0.5 because this value appears to be optimum for a wide range of networks and operations. However, for input units, the optimum probability of units' maintenance is close to 1 instead of 0.5.

Applying dropout to a neural network means sampling a "thin" network from it. The thin network contains all the units that remain after implementing the random deletion method. A neural network with n units can be considered as a set of 2^n thin neural networks. These networks have shared weights; therefore, the total number of parameters is still equal to $O(2^n)$ or may be lower than this value. If we show the state of training in any case, a new thin network is sampled and trained. Consequently, neural network training with the random deletion method can be considered as training a set of 2^n thin networks with extensive weight sharing where each thin network is rarely trained or not trained at all. At the same time, it is not possible to obtain the average predictions of thin models, the number of which is in exponential order. However, there is a simple approximate averaging method that works well in such a situation. The idea is to use a neural network without dropout during the test. The weights of this network are versions of the trained weights. If a unit with p probability is trained during training, the output weights of that unit are multiplied by p at the time of testing. This process ensures that for each hidden unit, the expected output is the same as the actual output at test time (given the distribution used to dilute the units during training). By performing this scalability method, the 2^n network with shared weights can be combined with a neural network that is used during the test. We found that training a network with a random deletion method and using this technique of approximate averaging at the time of testing leads to a lower generalization error[29-30,18].

3.2 Dropout model

Consider a neural network with L hidden layers. Suppose $l \in \{1, \dots, L\}$ to show the hidden layers of the network. Consider $z^{(l)}$ to represent the input vector to layer l and $y^{(l)}$ to show the output vector of l layer ($y^{(0)}=x$ is input). $W^{(l)}$ and $b^{(l)}$ are the weights and biases in l layer. The feed-forward operation of a standard neural network can be described as follows [for $l \in$ {0,…, L-1} and each i hidden layer]:

$$z_i^{(l+1)} = w_i^{(l+1)} y^l + b_i^{(l+1)},$$
$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right), \quad (1)$$

In the above equations, f can be any activation function and using the random deletion method, the feed-forward operation is as follows:

$$r_j^{(l)} \sim \text{Bernoulli(p)},$$
$$\tilde{y}^l = r^{(l)} * y^{(l)},$$
$$z_i^{(l+1)} = w_i^{(l+1)} \tilde{y}^l + b_i^{(l+1)}, \quad (2)$$
$$y_i^{(l+1)} = f\left(z_i^{(l+1)}\right)$$

Here, * refers to element-wise multiplication. For each l layer, $r^{(l)}$ is a vector of independent Bernoulli random variables, each with a probability of p equal to one. This vector is sampled and multiplied element-wise by the outputs of that layer ($y^{(l)}$) to produce the outputs of thin$\tilde{y}^l$ outputs. These outputs are then used as input to the next layer, and this process applies to each layer. The process can be considered as sampling a partial network from a larger network. In the learning process, the derivative of loss is back propagated into the partial network. At the time of testing, the weights are scaled as$W_{\text{test}}^{(l)} = pW^{(l)}$.

3.3 Learning random deletion networks

Random deletion neural networks can be trained using a stochastic gradient random method similar to standard neural networks. The only difference is that for each training group in mini-batch mode, we are actually training a thin network by diluting the units. Back propagation and feed-forward processes for training mode are performed only on this thin network. For training in each small group,

the gradients of each parameter are averaged. In any training mode where the parameter is not used, the value of the gradient of that parameter will be zero.

4. Feed-forward neural networks based on time series

These networks were actually created to process comet signals. In a typical neural network, all inputs and outputs are independent of each other, but in many cases, this idea can be very bad. For example, suppose you are trying to predict the next word in a sentence. If the network cannot learn the relationships between the words, it is certainly unable to predict the next word correctly.

Let's look at this type of network from another perspective; these networks have a type of memory that records information it has ever seen. In theory, it seems that these networks can record and use the information in a long sequence, but in practice, this is not the case and they are very limited, in that they only record information from a few steps ago. There are many types of time series, including time series neural networks. In this paper, RNN, LSTM, and NARX networks and a proposed network based on LSTM Encoder-Decoder will be examined.

4.1 NARX time series neural network

Conventional neural networks such as Feed-Forward neural networks focus only on the input layers and transmit the network inputs directly to the neurons in each layer, producing an output [31-33]. NARX neural networks are based on linear models and are a dynamic return network that, in addition to removing the features of the Feed Forward network model, use the network outputs as feedback to the input of the previous layers of the neural network. Also, in these networks, in addition to normal inputs, there are other inputs called delay inputs that are injected into the network. The time required to maintain the values of the previous inputs of the network x(t) and the output sequence is y(t);in these networks, to start work, first, the training data is loaded, and then with the delay lines specified for the inputs and the output sequence, if there are two delays for input and output, the neural network training will start from the third point, the inputs are applied in parallel to the network, and in fact, the output y(t) is one feedback and one input. Now, for the neural network considered m, the general equation of the NARX neural network is given below:

$$y(t) = \tau\, f(y(t-1), \dots, y(t-d_y), u(t-1), u(t-2), \dots, u(t-d_u)), \quad (D=1,2,\dots) \tag{3}$$

$$y(t+D) = \tau\, f(y(t), \dots, y(t-d_u), u(t), u(t-1), \dots, u(t-d_u)), \quad (D=1,2,\dots) \tag{4}$$

In this equation that is a nonlinear function, x(t) and y(t) are the input and output of the model at time t, and d(y) and d(u) are the delays for the input and output of the system, respectively; if the value of D is one, the prediction is one step ahead, and if the value of D is more than one, the prediction is several steps ahead.

In fact, the general idea of time series is that from a series of values specified in a time such as (t) to predict other points in future times (t + x), the Figure below shows the general representation of the neural network system (Figure 1).

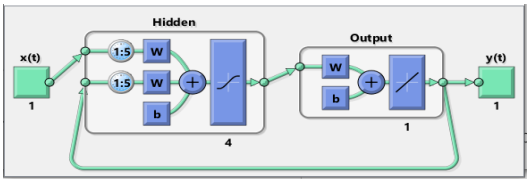

Figure 1. NARX neural network

4.2 Recurrent neural network (RNN)

In recurrent neural networks, for each sequential element, the same task is performed and the output of the current state depends on the previous states [34]. Because these networks retain the results of previous calculations and use them for future computations, they are also called memory networks. The simple architecture for RNN is shown in Figure 2. Here are some steps we can take to begin the process of preparation for mediation:

- Give input to the network (Input $_t$).
- Calculate the current state based on the previous state and the current input; in other words, calculate State$_s$ as follows:
  $\text{State}_t = f(\text{State}_{t-1}, \text{Input}_t)$ (5)
- For the next time step, State t becomes State $_{t-1}$.
- At the request of the problem, perform the required number of time steps and then combine the information of all the previous cases.
- Calculate Output$_t$ using the final current state:
  $\text{Output}_t = W_{output} * \text{State}_t$ (6)
- Here, W $_{output}$ is the weight in the output neurons.
- Calculate the error by finding the difference between the actual and predicted output.
- Update weights by error back propagation on the network.

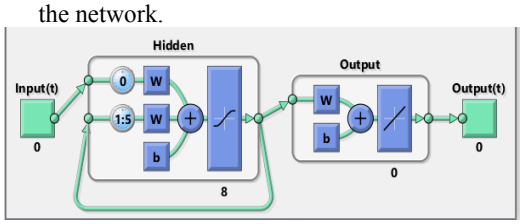

Figure 2. Recurrent Neural Network

4.3 LSTM

Research has focused on applying dropout methods to recurrent connections. Applying standard dropout to these connections results in poor performance since the noise caused by dropout at each time step prevents the network from retaining long-term memory. However, methods that are specialized for recurrent layers have proved successful and are commonly used in practice. Generally speaking, they apply dropout to recurrent connections in a way that can still preserve long-term memory [35-36]. Research into dropout in recurrent neural networks (RNNs) has focused on long short-term memory (LSTM) networks, although some proposed methods can be applied to RNNs in general. The following is a typical definition of an LSTM cell, although variations exist in this regard. For an input x_t at time t, input, forget, and output, gate signals are defined as:

$$i_t = \sigma(W_i x_t + U_i h_{t-1}) \quad (7)$$

$$f_t = \sigma\big(W_f x_t + U_f h_{t-1}\big), \quad (8)$$

And

$$o_t = \sigma(W_o x_t + U_o h_{t-1}) \quad (9)$$

Respectively, the cell state is defined as:

$$c_t = f_t o c_{t-1} + i_t o g_t, \quad (10)$$

Where

$$g_t = \tanh(W_g x_t + U_g h_{t-1}) \quad (11)$$

The hidden state, which is the layer's output, is defined as:

$$h_t = o_t o \tanh(c_t) \quad (12)$$

W and U matrices represent learned weights, σ(·) shows a sigmoid activation function, and σ(·) and tan h(·) are applied element-wise.

5. Proposed approach

In this section, the proposed approach in subsection 5 is presented. First, LSTM Encoder-Decoder is explained in subsection 5.1, and Dropout in LSTM Cells is presented in subsection 5.2, and in 5.3, the proposed method is described in detail.

5.1 LSTM Encoder-Decoder Architecture

The LSTM Encoder-Decoder architecture was first introduced for machine translation.

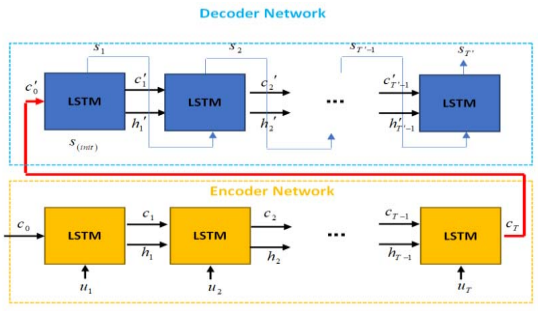

Figure 3. LSTM Encoder-Decoder

It has the ability to read and generate a sequence of arbitrary length as illustrated in Figure.3. The architecture employs two LSTM networks called the encoder and decoder. The encoder processes the input sequence $u_1,\ldots, u_T$ of the length T and produces the summary of the past input sequence through the cell state vector ct. After T times of recurrent updates from (1) through (5), the encoder summarizes the whole input sequence into the final cell state vector c_T. Then, the encoder passes c_T to the decoder so that the decoder uses it as the initial cell state (i.e., $c'_0 = c_T$) for the sequence generation. The decoding step is initiated with a dummy input s(init). The decoder recurrently generates the output sequence $s_1,\ldots, s_{T'}$ of the length T'. In every update, the decoder feeds the output s_{t-1} obtained in the previous update to the input for the current update. Note that the output of the decoder is derived by applying the affine transformation followed by the function that suits the specific tasks (e.g. Softmax function for classification task). Basically, the LSTM Encoder-Decoder aims to model the conditional probability of the output sequence given the input sequence, i.e. $p(s_1,\ldots, s_{T0} |u_1,\ldots, u_T)$. The encoder provides the summary of the input sequence $u_1,\ldots, u_T$ through the LSTM cell state c_T. Given the encoder cell state c_T, the conditional probability is approximated to

$$p(s_1,,\ldots.,s_{T'}|u_1\ldots,u_T) \approx \prod_{t=1}^{T'} P(s_1|C_T,s_1,..,s_{t-1}) \quad (13)$$

The decoder successively produces the probability distribution of $P(s_1|\acute{c}_{t-1}, s_{t-1})$ given the decoder cell state $\acute{c}_{t-1}$ and the (t-1)the sample of the output sequence s_{t-1}:

$$p(s_1,,\ldots.,s_{T'}|u_1\ldots,u_T) \approx \prod_{t=1}^{T'} P(s_1|\acute{c}_{t-1},s_{t-1}) \quad (14)$$

Unfortunately, the decoder does not know the true value of the previous output sample. Hence, in every decoding step, the decoder decides on s_t based on the probability distribution $p(s_t|\acute{c}_{t-1}, s_{t-1})$ obtained from the decoder output and uses the tentative decision for the next update of the decoder state.

5.2 Dropout in LSTM Cells

In this section, we applied dropout in LSTM and LSTM Encoder-Decoder, both of which use the LSTM cell.

The key change is to generate a dropout mask for each input sequence and then keep it.

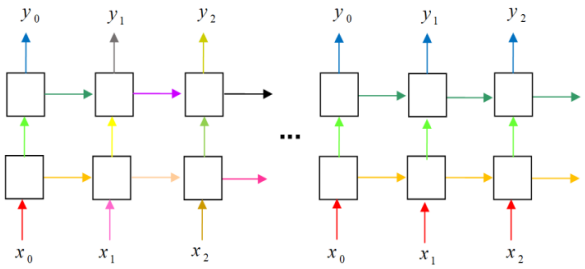

Figure 4. Comparison of per-step (left) versus per-sequence (right) sampling of dropout masks on an unrolled RNN. Horizontal connections are recurrent while vertical connections are feed-forward. Different colors represent various dropout masks applied to the corresponding connection.

The same is true at every time step. This varies from the naive way of applying dropout to RNNs, which would generate new dropout masks for each input sample regardless of which time sequence it was from. Generating masks on a per-sequence basis means that the elements in the network's hidden state that are not dropped will persist throughout the entire sequence without ever being affected by dropout, which allows the network to maintain long-term memory. The difference between per-step and per-sequence masks on an unrolled RNN is illustrated in Figure 4. In particular, the authors propose applying dropout to the hidden cell state. Hence, the only change from the original LSTM definition is the equation for ct, which becomes as follows:

$$c_t = mo(f_t oc_{t-1} + i_t og_t), \quad m_i \sim Bernoulli(1-p), \tag{15}$$

Various other proposed methods also use per-sequence dropout mask sampling on recurrent connections to help preserve long-term memory. Variational RNN dropout, which was presented in 2016, is one such method, but it operates in a way that is theoretically justified in terms of a Bayesian interpretation of RNN dropout. The authors show that if dropout is seen as a variational Monte Carlo approximation to a Bayesian posterior, then the natural way to apply it to recurrent layers is to generate a dropout mask that zeroes out both feed forward and recurrent connections for each training sequence but to keep the same mask for each time step in the sequence. This is similar to RNN drop in that masks are generated on a persequence basis, but the derivation leads to dropout being applied at a different point in the LSTM cell. Formally, the equations for it, ft, o_t, and g_t take the following forms:

$$i_t = \sigma\big(W_i(x_t om_x) + U_i(h_{t-1} om_h)\big) \tag{16}$$

$$f_t = \sigma\Big(W_f(x_t om_x) + U_f(h_{t-1} om_h)\Big) \tag{17}$$

$$o_t = \sigma\big(W_o(x_t om_x) + U_o(h_{t-1} om_h)\big) \tag{18}$$

$$g_t = tanh\Big(W_g(x_t om_x) + U_g(h_{t-1} om_h)\Big) \tag{19}$$

$$m_{x,i}, m_{h,i} \sim Bernoulli(1-p) \tag{20}$$

With the equations for c_t and h remaining the same as in the original LSTM. This dropout method has become one of the most widespread techniques for regularizing RNNs. In training, these LSTM cells use the following equations for it, namely f_t, o_t, and g_t, otherwise following the basic LSTM formulation given below.

$$i_t = \sigma(W_i x_t (U_i oM) h_{t-1}) \tag{21}$$

$$f_t = \sigma\big(W_f x_t + (U_f oM) h_{t-1}\big) \tag{22}$$

$$f_t = \sigma(W_o x_t + (U_o oM) h_{t-1}) \tag{23}$$

$$g_t = tanh\big(W_g x_t + (U_g oM) h_{t-1}\big) \tag{24}$$

$$M_{ij} \sim Bernoulli(1-p) \tag{25}$$

This approach allowed the authors to achieve results on language modeling benchmarks that were state-of-the-art at the time. Recurrent dropout is an alternative approach that can preserve memory in an LSTM while still generating different dropout masks for each input sample as in standard dropout. This is done by only applying dropout to the part of the RNN that updates the hidden state and not the state itself. So, if an element is dropped, then it simply does not contribute to network memory, rather than erasing the hidden state. For an LSTM, the equations are the same as in the original LSTM except that the equation for ct becomes as follows:

$$c_t = f_t oc_{t-1} + i_t og_t om_t, \; m_{t,i} \sim Bernoulli(1-p). \tag{26}$$

5.3 Model Design

The proposed architectural model is shown in Figure 5. This network consists of two main components: i) an Encoder-Decoder framework that captures the inherent pattern in the time series, which is learned during the pre-training step, and (ii) a prediction network that takes input from both the learned embeddings from Encoder-Decoder. We discuss the two components in more detail below.

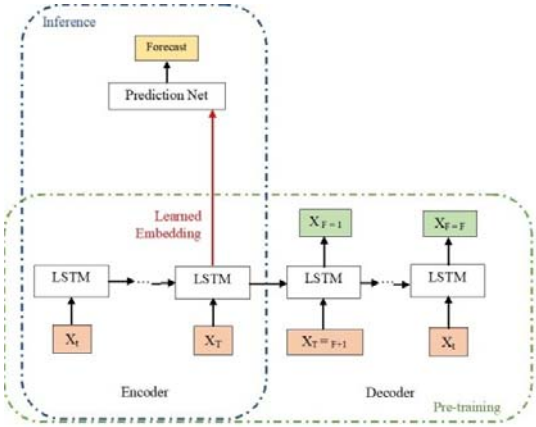

Figure 5. Neural network architecture, with a pre-training phase using an LSTM Encoder-Decoder

5.3.1 Encoder-decoder

Before fitting the prediction model, we conduct a pre-training step to fit an encoder that can extract useful and representative embeddings from a time series. The goals are to ensure that (i) the learned embedding provides useful features for prediction and (ii) unusual input can be captured in the embedded space, which will get further propagated to the prediction network in the next step. Here, we use an Encoder-Decoder framework with two-layer LSTM cells.

Specifically, given a univariate time series {xt}t, the encoder reads in the first T timestamps {x1, ..., xT } and constructs a fixed-dimensional embedding state. After that, from this embedding state, the decoder constructs the following F timestamps {xT +1, ..., xT +F } with guidance from {xT −F +1, ..., xT } (Figure 5, bottom panel). The intuition is that in order to construct the next few timestamps, the embedding state must extract representative and meaningful features from the input time series. This design is inspired by the success

of video representation learning using a similar architecture [Srivastava et al., 2015].

5.3.2 Prediction network

After the Encoder-Decoder is pre-trained, it is treated as an intelligent feature-extraction black box. Specifically, the last LSTM cell states of the encoder are extracted as learned embedding. Then, a prediction network is trained to forecast the next one or more timestamps using the learned embedding as features. In the scenario where external features are available, these can be concatenated to the embedding vector and passed together to the final prediction network. Here, we use a multi-layer perceptron as the prediction network.

6. Evaluation

This section contains a number of subsections. We first implemented our method on a data set taken from the Launch Abort System (LAS) subsystem. We then implemented our results in two other datasets introduced in the papers [Tohma et al., 1991; Tohma et al., 1989].

Subsequently, for better comparison, in addition to LSTM Encoder-Decoder (Proposed Method), we also implemented this dataset on NARX and RNN recurrent networks.

These datasets are all time-series identification systems. It should be noted that the input is the software test time and the output is the cumulative number of faults at that time (Figure 6).

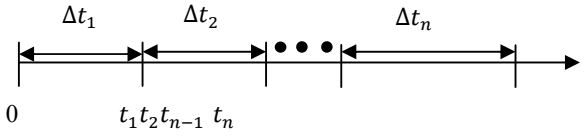

Figure 6. Failure process in software

6.1 Implementation on-air/space application

We applied the results of our approach to part of a real CPS known as Launch Abort System (LAS), the architecture of which is shown in Figure 7. As indicated, the tree construction of a space vehicle Launch Abort System (LAS) has functional dependencies among the elements. This system consists of two main subsystems, which include the Emergency Detection System (EDS) as a module of the air/space vehicle system and the Launch Escape System (LES). EDS is used for detecting the abnormal and emergency conditions in flying vehicles that are designed to treat mechanical and electrical failures in each of the propulsion, electrical, and control portions. While the EDS system detects an unusual and faulty condition in one of the main parts of the LES system (i.e., electrical power failure, structural failure, guidance, and control faulty, etc.), it is sensed and transmitted through a signal to trigger the FDEP gate of LES system. Generally, the launch escape system consists of a launch escape motor, pitch control motor, tower jettison motor, landing parachute, and the Master Event Sequencing Controller Subsystems (MESC). According to the launch and flight regimes and based on the flight altitude, the operation time of each motor-based subsystem is different. A master event sequence controller on LES is an intelligent standalone microprocessor-based system, which monitors external inputs and controls the time and sequence of the event's changes. MESCs are therefore utilized as a prioritizing tool to dictate the occurrence of events.

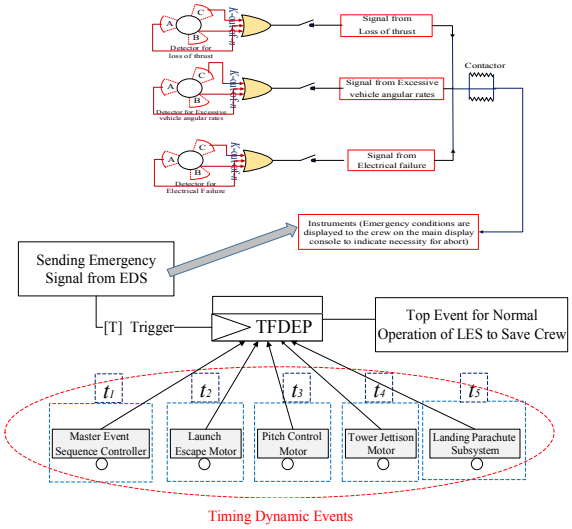

Figure 7. Tree construction with the dependency on LAS

6.2 Other Datasets

This section compares the proposed model with NN and other parametric models. We utilize the model comparison results to compare our results with those from previous studies. We employ two fault data sets to compare the prediction performance of all the models in the present study. These datasets have more data rather than dataset 1.

Dataset1: The first fault data set (DS1) was collected from a real-time control application with approximately 870,000 code lines [37].

Datasat2: A total of 481 faults were detected from a monitoring and real-time control system with approximately 200,000 code lines for 111 days. The second fault date set (DS2) was collected from [38].

6.3 Assessment methods

The performance of software reliability models is highly dependent on the estimation of model parameters. In this paper, to assess our models, we evaluate MSE errors using two different optimizers, namely Adam and SGD on three

datasets. The MSEs are reported after 150 and 1000 iterations. As can be seen in Tables 1-2 and Figures 8-15, the LSTM Encoder-Decoder model (proposed method) outperforms rather another RNN-based model (especially in datasets with more data(dataset 2,3)). Furthermore, introducing dropouts can even further improve the results that were obtained earlier. Therefore, the LSTM Encoder-Decoder dropout achieves the lowest error rates, especially with the Adam optimizer dropout method (proposed method with dropout(Figures 11-13)).

Table 1. Validation Metrics for confirmed cases of LAS using **MSE** with 150 Epochs

	LAS		Dataset1		Dataset2	
Optimizer	**Adam**	**SGD**	**Adam**	**SGD**	**Adam**	**SGD**
LSTM Encoder-Decoder	0.0015	0.05	9.2260e-05	0.00099	9.4631e-06	0.0030
LSTM Encoder-Decoder+ Drop Out	0.0010	0.053	0.00011	0.00076	7.3260e-06	0.00072
LSTM	0.0128	0.0254	14.0857	1.5279e+03	7.5597	22.1092
LSTM+Dropout	0.0125	0.1173	13.9176	2.5611	9.7327	13.9842
NARX	1.85e-25	4.16e-22	0.0496	9.32e+03	63.5	0.0124
NARX +Dropout	1.83e-29	5.26e-30	13.8	86.1	0.0136	65.7
RNN	3.37e-29	1.02e-19	1.08e+3	0.000279	5.74e-5	1.49e-27
RNN+Dropout	1.88e-27	7.51e-29	6.72e+03	0.0155	3.19e+03	679

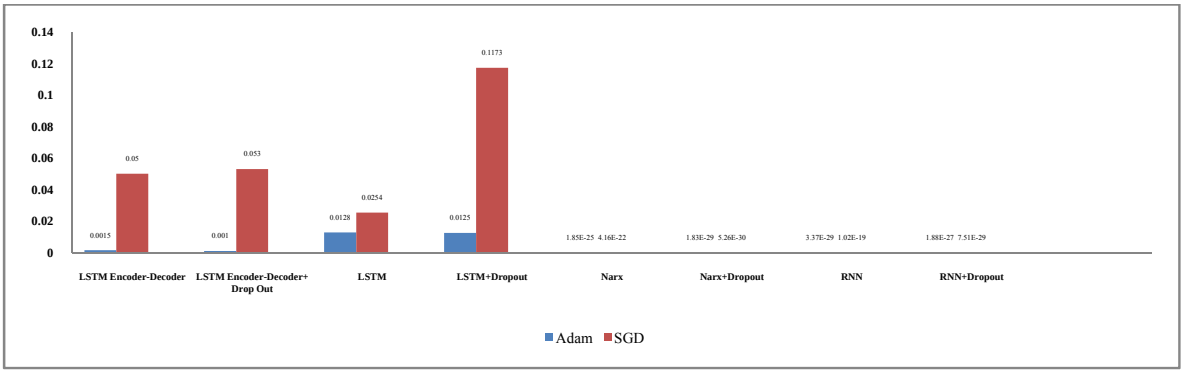

Figure 8. Validation Metrics for LAS dataset using **MSE** with 150 Epochs

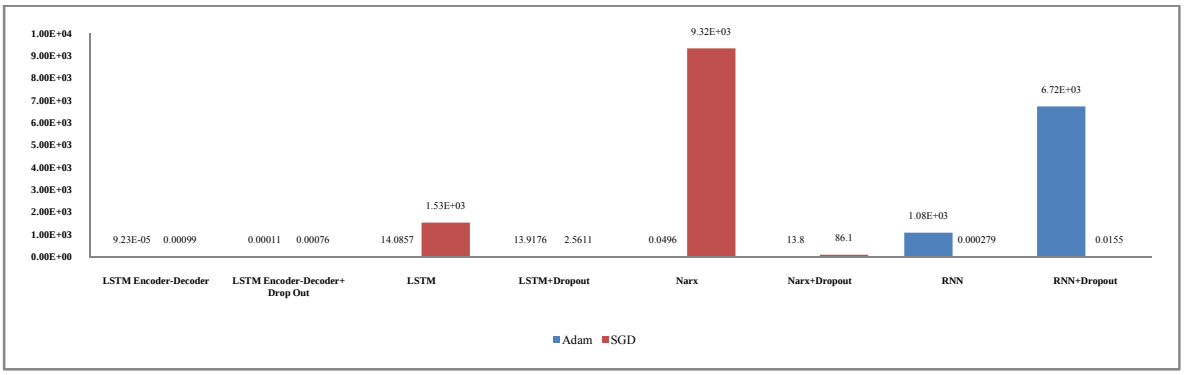

Figure 9. Validation Metrics for dataset1 using **MSE** with 150 Epochs

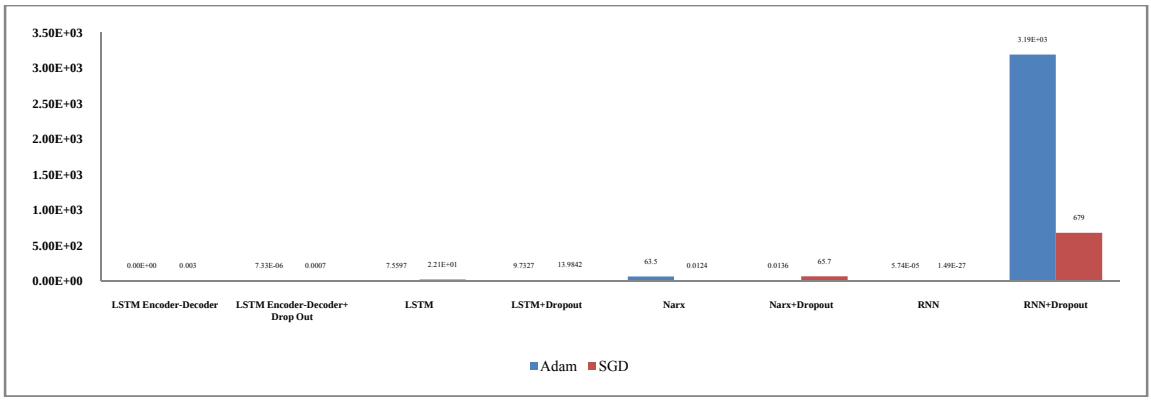

Figure 10. Validation Metrics for dataset2 using **MSE** with 150 Epochs

Table 2. Validation Metrics for confirmed cases LAS using MSE with 1000 Epochs

Dataset	LAS		Dataset1		Dataset2	
Optimizer	**Adam**	**SGD**	**Adam**	**SGD**	**Adam**	**SGD**
LSTM Encoder-Decoder	0.00075	0.04	1.34155e-05	1.3415e-05	6.6070e-05	4.6826e-05
LSTM Encoder-Decoder+ Drop Out	0.00069	0.04	0.00021	0.0002	1.6009e-05	0.00011
LSTM	9.1421e-04	0.0272	889.9926	1.0842e+04	86.5335	1.4553e+04
LSTM+Dropout	0.0077	0.1069	7.3391	7.1412	9.6365	27.4822
NARX	3.17e-27	2.61e-27	0.00057	226	50	177
NARX +Dropout	8.59	9.68e-25	154	101	0.00767	154
RNN	3.21e-25	2.05e-26	2.07e-25	5.18e+03	3.2e-25	2.53
RNN+Dropout	669	2.31e-30	1.25e-24	0.751	0.0599	3.68

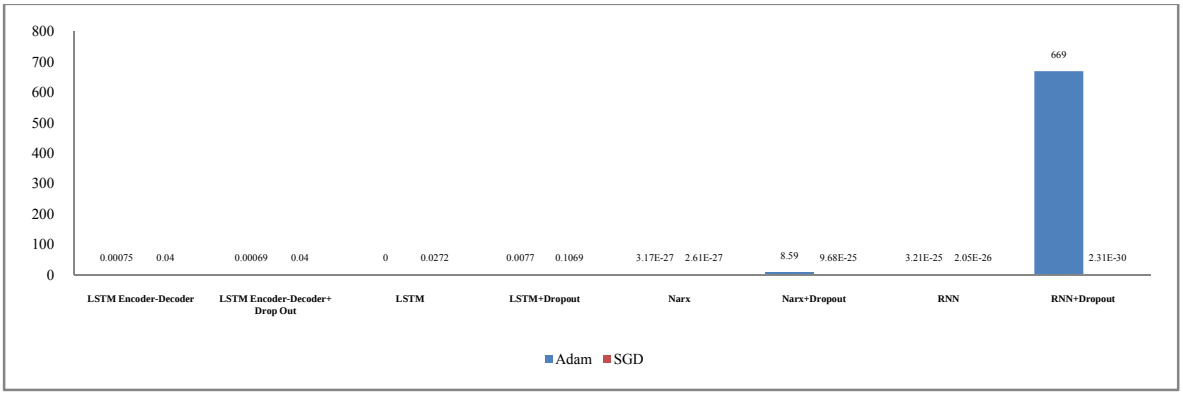

Figure 11. Validation Metrics for LAS using MSE with 1000 Epochs

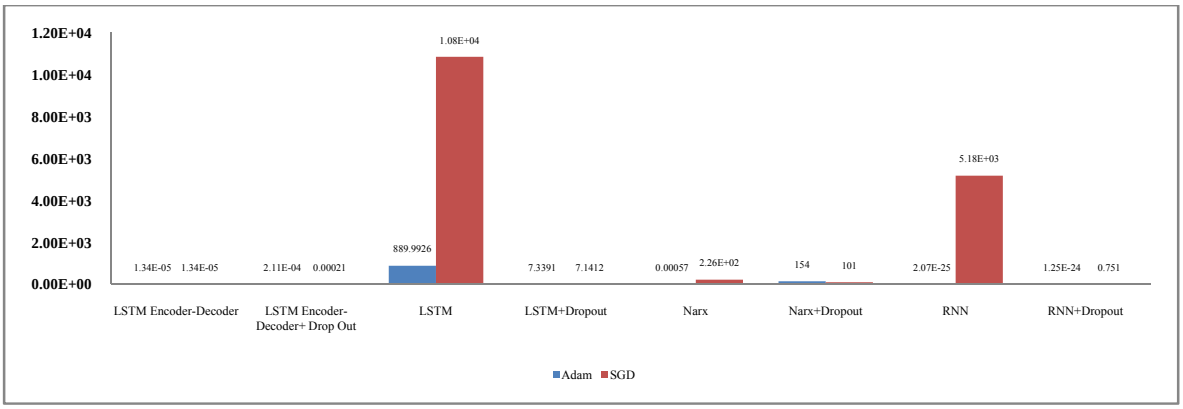

Figure 12. Validation Metrics for dataset1 using MSE with 1000 Epochs

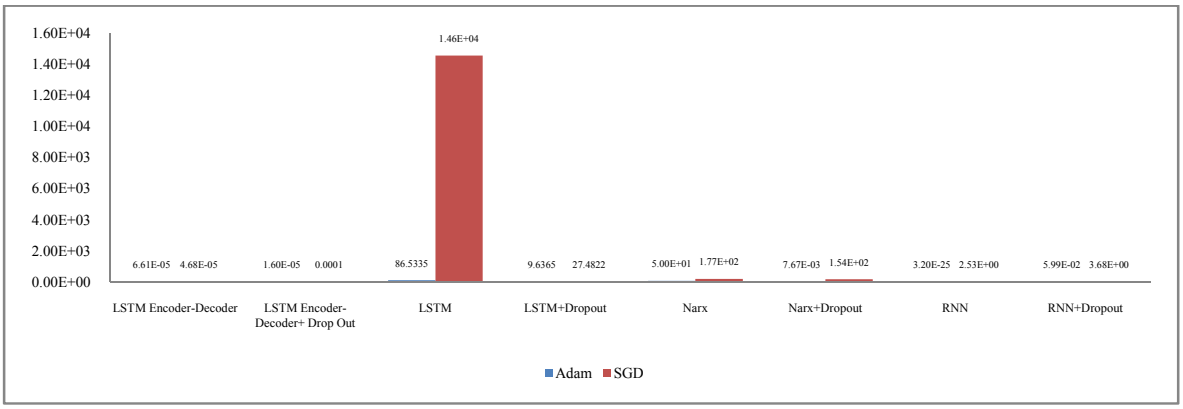

Figure 13. Validation Metrics for dataset2 using MSE with 1000 Epochs

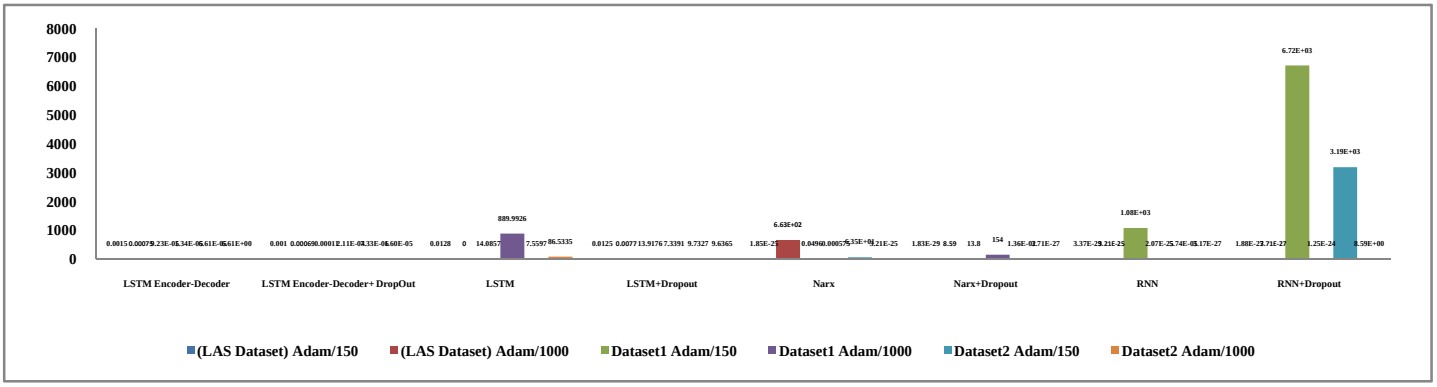

Figure 14. Validation Metrics for confirmed cases of LAS based on Adam classification

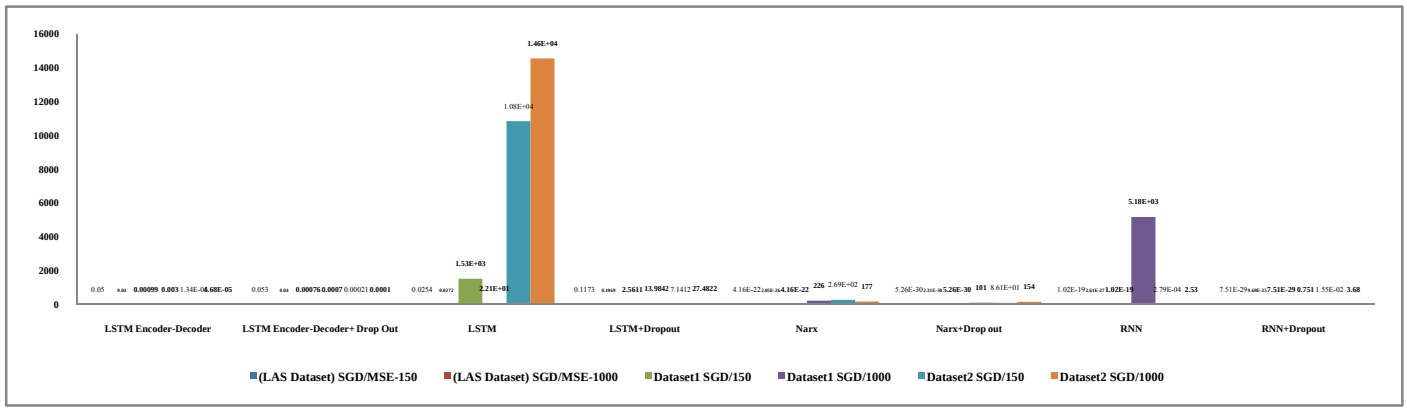

Figure 15. Validation Metrics for confirmed cases of LAS based on SGD classification

7. Conclusion

The use of non-parametric methods (e.g. deep recurrent neural networks) in software reliability prediction applications has shown great promise. In this paper, we employed RNN-based networks to predict the reliability of software and presented a method based on Encoder-Decoder. A major issue that needs to be considered is that of overfitting. We show that the use of dropout in the training phase is highly efficacious in this regard. Simulations on three datasets using two different optimizers indicate that the best results in terms of MSE pertain to the LSTM Encoder-Decoder model with dropout (proposed method), especially in data sets with more data (dataset2, dataset 3). As a future direction, we may investigate the level of uncertainty involved in the predictions made by such networks using Bayesian approaches.

8. References

[1] H. Soltanali, Rohani, A., Abbaspour-Fard, M. H., &J. T. Farinha, "A comparative study of statistical and soft computing techniques for reliability prediction of automotive manufacturing", *Applied Soft Computing*, vol. 98, 106738. 2021.

[2] S. Oveisi, , & R. Ravanmehr, "SFTA-Based Approach for Safety/Reliability Analysis of Operational Use-Cases in Cyber-Physical Systems", *Journal of Computing and Information Science in Engineering*, vol. 17 no.3, 2017.

[3] S. Oveisi, M. A. Farsi, M. Nadjafi, & A. Moeini, "A New Approach to Promote Safety in the Software Life Cycle," *Journal of Computer & Robotics*, vol. 12, no.1, pp.77-91,2019.

[4] S. Oveisi, M. A. Farsi, M. Nadjafi, A. Moeini, & M. habankhah," Design Software Failure Mode and Effect Analysis using Fuzzy TOPSIS Based on Fuzzy Entropy," *Journal of Advances in Computer Engineering and Technology*, vol.6, no. 3, pp.171-180,2020.

[5] W. D. Van Driel, J. W. Bikker, & M. Tijink, "Prediction of software reliability," *Microelectronics Reliability*, vol. 119, 114074,2021.

[6] S. Oveisi, &M. A. Farsi, "Software safety analysis with UML-Based SRBD and fuzzy VIKOR-Based FMEA, "*International Journal of Reliability, Risk and Safety: Theory and Application*, vol.1,no.1,pp.35-44, 2018.

[7] G. Aggarwal, V. K. Gupta, "Software reliability growth model," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol.4, no.1, 2014.

[8] A. Jaiswal, & R. Malhotra, "Software reliability prediction using machine learning techniques," *International Journal of System Assurance Engineering and Management*, vol.9,no.1, pp.230-244,2018.

[9] K. Sahu, & R. K. Srivastava, "Revisiting software reliability, "in *Data Management, Analytics and Innovation. Advances in Intelligent Systems and Computing*, Balas, V., Sharma, N., Chakrabarti, A. (eds), vol. 808. Springer, Singapore.

[10] J. Wang, & C. Zhang, "Software reliability prediction using a deep learning model based on the RNN encoder–decoder, "*Reliability Engineering & System Safety*, vol.170, pp.73-82, 2018.

[11] A. Borovykh, S. Bohte, & C. W. Oosterlee, "Conditional time series prediction with convolutional neural networks, "*Journal of Computational Finance*, vol.22, no.4.2018.

[12] Z. Che, S. Purushotham, K. Cho, D. Sontag, &Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific reports*, vol.8,no.1, pp.1-12, 2018.

[13] Wang, H., Yang, Z., Yu, Q., Hong, T., & Lin, X. (2018). Online reliability time series prediction via convolutional neural network and long short term memory for service-oriented systems. Knowledge-Based Systems, 159, 132-147.

[14] K. L. H. Nguyen, "Uncertainty in Recurrent Neural Network with Dropout", Master Thesis. Department of Computer, Communication and Information Sciences, AALTO University, 2020.

[15] Y. Gal, & Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," *Advances in neural information processing systems*, vol.29, pp.1019-1027, 2016.

[16] A. Labach, H. Salehinejad, &, S. Valaee "Survey of dropout methods for deep neural networks" arXiv preprint arXiv:1904.13310.2019.

[17] F. Di Nunno, & F.Granata, "Groundwater level prediction in Apulia region (Southern Italy) using NARX neural network" *Environmental Research*, vol.190, 110062,2020.

[18] Y.Gal & Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," In International conference on machine learning, 2016, pp. 1050-1059, PMLR..

[19] Y. Jeong, S. Kim, & K. Yi, "Surround vehicle motion prediction using LSTM-RNN for motion planning of autonomous vehicles at multi-lane turn intersections, "*IEEE Open Journal of Intelligent Transportation Systems*, vol.1, pp.2-14, 2020.

[20] Z. Lv, , Xu, J., Zheng, K., Yin, H., Zhao, P., & Zhou, X. "Lc-rnn: A deep learning model for traffic speed prediction" . *IJCAI* ,pp. 3470-3476,2018.

[21] N. Srivastava, "Improving neural networks with dropout"., MSc thesis, University of Toronto, 2013.

[22] I. Sutskever, O. Vinyals, &Q. V. Le, "Sequence to sequence learning with neural networks, "*Advances in neural information processing systems*, pp. 3104-3112, 2014.

[23] R. Pascanu, C. Gulcehre, K. Cho, &Y. Bengio, "How to construct deep recurrent neural networks". arXiv preprint arXiv:1312.6026.2013.

[24] H. Wang, Z. Yang, Q. Yu, T. Hong, &X. Lin, "Online reliability time series prediction via convolutional neural network and long short term memory for service-oriented systems," *Knowledge-Based Systems*, vol. 159, pp.132-147, 2018.

[25] M. K. Bhuyan, D. P. Mohapatra, & S. Sethi, "Software Reliability Prediction using Fuzzy Min-Max Algorithm and Recurrent Neural Network Approach," *International Journal of Electrical & Computer Engineering*, vol. 6no. 4. Pp.2088-8708, 2016.

[26] J. Wang, & C. Zhang, "Software reliability prediction using a deep learning model based on the RNN encoder–decoder," *Reliability Engineering & System Safety*, vol.170, pp.73-82, 2018.

[27] R. Mittelman, "Time-series modeling with undecimated fully convolutional neural networks," arXiv preprint arXiv:1508.00317.2015.

[28] M. Binkowski, G. Marti, , & P. Donnat, "Autoregressive convolutional neural networks for asynchronous time series," In International Conference on Machine Learning pp. 580-589, PMLR. 2018.

[29] S. N.rivastava, G. Hinton, A. Krizhevsky, I. Sutskever, &R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of machine learning research*, vol.15,no.1,pp. 1929-1958,2014.

[30] A. Labach, H. Salehinejad, & S.Valaee, "Survey of dropout methods for deep neural networks," arXiv preprint arXiv:1904.13310,2019.

[31] F. Di Nunno, & F. Granata, "Groundwater level prediction in Apulia region (Southern Italy) using NARX neural network," *Environmental Research*, vol.190,110062.2020.

[32] F.Di Nunno, G. de Marinis, R. Gargano, & F. Granata, " Tide prediction in the Venice Lagoon using Nonlinear Autoregressive Exogenous (NARX) neural network," *Water*, vol.13,no.9, 1173, 2021.

[33] F. Di Nunno, F. Granata, R. Gargano, & de Marinis, G. "Forecasting of extreme storm tide events using NARX neural network-based models," *Atmosphere*, vol.12, no.4, 512,2021.

[34] S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V.K. Menon, &K. P. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," In *2017 international conference on advances in computing, communications and informatics (icacci)* (pp. 1643-1647). September 2017.

[35] N. Srivastava, E. Mansimov, & R. Salakhudinov, "Unsupervised learning of video representations using lstms". In International conference on machine learning pp. 843-852, PMLR.,2015

[36] R. Wang, F. Yan, J. Lu, &W. Y. Yang, "COVID-19 Trend Forecasting by Using Dropout-LSTM Model," *Dianzi Keji Daxue Xuebao/Journal of the University of Electronic Science and Technology of China*, vol.50,no.3, pp. 414-421, 2021.

[37] Y. Tohma, H. Yamano, M. Ohba, & R. Jacoby, "Parameter estimation of the hyper-geometric distribution model for real test/debug data". In Proceedings. 1991 International Symposium on Software Reliability Engineering (pp. 28-29). IEEE Computer Society 1991.

[38] Y. Tohma, K. Tokunaga, S. Nagase, &Y. Murata, " Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution, "*IEEE transactions on software engineering*, vol. 15,no. 3, pp. 345-355,1998.