IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 8, 2022, accepted March 2, 2022, date of publication March 8, 2022, date of current version March 17, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3157306

An Interpretation of Long Short-Term Memory Recurrent Neural Network for Approximating Roots of Polynomials

MADIHA BUKHSH[1], MUHAMMAD SAQIB ALI[2], MUHAMMAD USMAN ASHRAF[3], KHALID ALSUBHI[4], AND WEIQIU CHEN[1], (Member, IEEE)

[1]Key Laboratory of Soft Machines and Smart Devices of Zhejiang Province, Department of Engineering Mechanics, Zhejiang University, Hangzhou 310027, China
[2]Department of Electrical and Computer Engineering, COMSATS University Islamabad (CUI), Islamabad 45550, Pakistan
[3]Department of Computer Science, Government College Women University, Sialkot, Sialkot 53310, Pakistan
[4]Department of Computer Science, King Abdulaziz University, Jeddah 21589, Saudi Arabia

Corresponding authors: Madiha Bukhsh (madiha@zju.edu.cn) and Muhammad Usman Ashraf (usman.ashraf@gcwus.edu.pk)

This work was supported in part by the Deanship of Scientific Research (DSR), King Abdulaziz University, Jeddah, under Grant D-326-611-1442; in part by DSR; and in part by the Project funded by the Shenzhen Scientific and Technological Fund for Research and Development, China, under Grant 2021Szvup152.

ABSTRACT This paper aims to present a flexible method for interpreting the Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) for the relational structure between the roots and the coefficients of a polynomial. A database is first developed for randomly selected inputs based on the degrees of the univariate polynomial which is then used to approximate the polynomial roots through the proposed LSTM-RNN model. Furthermore, an adaptive learning optimization algorithm is used specifically to update the network weights iteratively based on training deep neural networks data. Thus, the method can exploit the ability to find the individual learning rates for each variable through adaptive learning rate strategies to effectively prevent the weights from fluctuating in a wide spectrum. Finally, several experimental results are performed which shows that the proposed LSTM-RNN model can be used as an alternative approach to compute an approximation of each root for a given polynomial. Furthermore, the results are compared with the conventional feedforward neural network based artificial neural network model. The results clearly demonstrate the superiority of the proposed LSTM-RNN model for roots approximation in terms of accuracy, mean square error and faster convergence.

INDEX TERMS Long short-term memory, recurrent neural network, deep neural network, adaptive moment estimation algorithm, error cost function.

I. INTRODUCTION

Finding the roots (zeros) of a polynomial is a key problem in a variety of scientific and technical fields. There are several traditional iterative methods for determining the roots of a polynomial, e.g., Newton's method, Bisection method, Durand-Kerner (D-K) method and Laguerre's method, etc. [1], [2]. However, some common problems associated with these conventional methods are: (1) root leaping may occur, resulting in failure to achieve the desired result, as well as an inflection point; (2) choosing an estimate that is close to the root may require several iterations, leading to a slow convergence; and (3) the calculation of the first or second-order derivatives since being computationally intensive and therefore is not always possible.

To overcome the limitations of the conventional approaches, a two-layer feedforward neural network (FNN), as one of the neural networks (NN), was proposed to do polynomial factorization with two or even more variables [3], [4]. It is shown that the NN can adequately resolve the computational problems related to polynomial root searching. Also, the classical method for backpropagation algorithm (BPA) used in FNN with, nevertheless, gradient descent shows a slow convergence [5], [6]; thus, its use in NN computing is drastically limited. It is found that the BPA needs a lot of time for convergence unless the initial network synapse

The associate editor coordinating the review of this manuscript and approving it for publication was Hao Luo.

weights corresponding to the roots are selected properly. In 1997, Perantonis *et al.* initiated the use of constrained learning (CL) methodology to train BP networks and factorize two-dimensional polynomials by incorporating prior information from problems and the parameters for network structure learning [7]. In 2001, Huang, based on the Σ-Π structured NN [8], [9] proposed an artificial neural network (ANN) and included a priori information about the connections between the roots and the coefficients to discover a particular polynomial real and complex root [10]. In [11], Freitas and colleagues proposed a FNN based ANN technique for approximating roots of a given polynomial. Although the suggested ANN approach does not surpass the accuracy of the traditional iterative methods, the results are encouraging and show the effectiveness of the NN techniques for approximating the polynomial roots (complex or real). Furthermore, it is commonly known that by using flexible parallel architectures in NNs, we may obtain all roots simultaneously and in parallel, especially when the computations are performed on a parallel-computing machine. On the other hand, most nonlinear numerical algorithms can only identify one root at a time, thus increasing the number of processors will not speed up the process. As a result, numerical methods are significantly slower than the NN methods in terms of speed [10].

In the aforementioned literature survey, although slightly compromising the accuracy as compared to the traditional iterative methods which need larger processing time, the NN techniques show their potential and significance and can be an alternate way for finding the roots of polynomials with fewer effort. Therefore, this paper proposes a more advanced deep neural network (DNN) technique, namely the long short-term memory recurrent neural network (LSTM-RNN), for approximating the roots of a univariate polynomial. The purpose is to tackle the limitation of the conventional NN techniques such as FNN based NN models which do not make use of the initial interpretation capacity of NN for approximating the roots of a given polynomial. Furthermore, two common drawbacks of FNN based NN are: (1) falling into local minimum; and (2) the slow convergence makes the fully connected FNN inefficient to train and tend to overfit the model [12], [13]. On the other hand, to tackle such problems, the LSTM-RNN has become a very ardent research topic over a few years and which was first developed by Hochreiter and Schmidhuber in 1997 [14]. Several applications, such as neural computation and time series forecasting, filter design, unauthorized broadcasting identification, quality of transmission estimation etc. [15]–[20], are examples where the LSTM-RNN models have been successfully applied.

The LSTM-RNN based DNN technique can impose a confined relationship between the roots and the coefficients of a given polynomial. Also, the methodology can effectively train and validate the NN model. Similarly, how the datasets of coefficients and the roots of a given nth order polynomial can be generated and then the roots are approximately validating through the LSTM-RNN model are the research focus of this paper. LSTM-RNN structure mainly depends on layers, which consist of a set of recurrently connected blocks, known as memory blocks. These blocks can be called as a differentiable model, each one containing one or more recurrently connected memory cells and three multiplicative units: (1) the input gate; (2) the output gate; and (3) the forget gate. The particular gates provide the cells with continuous analogs of writing, reading and resetting operations. In addition, an error cost function is coupled with a constrained condition to alleviate the weight fluctuations in a wide range [21]. Besides, momentum is added to the learning algorithm to speed up the convergence [22].

The following is the organization of the rest of the paper. Section 2 presents the fundamental concept of univariate polynomial roots, discusses the LSTM-RNN model including the error cost function, and describes the optimizer based on the adaptive moment estimation (ADAM) algorithm and parameter learning. Section 3 shows the numerically experimental results with the discussions. Finally, Section 4 sets out several concluding remarks and directions for the future research.

II. METHODOLGY

A. nth-ORDER ARBITRARY POLYNOMIAL

The main focus of this study is to compute approximations of the roots of an n^{th} degree univariate polynomial based on its coefficients $a_i (i = 1, 2, \ldots, n)$. Hence, without loss of generality, four cases of $n = 5$, 10, 15 and 20 are considered in this study. A given nth-order polynomial $f(z)$ can be described as [6]:

$$\begin{aligned} f(z) &= a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots . a_{n-1} z + a_n \\ &= \sum_{k=0}^{n} a_{n-k} z^k \end{aligned} \tag{1}$$

where $n \geq 2$, $a_0 \neq 0$ and is usually taken to be 1. Suppose that there exist n approximate real or complex roots of $f(z)$. Then (1) can be factorized as,

$$\begin{aligned} f(z) &= a_0 z^n + a_1 z^{n-1} + a_2 z^{n-2} + \ldots . a_{n-1} z + a_n \\ &\approx \prod_{i=1}^{n} (z - w_i) \end{aligned} \tag{2}$$

where $w_i = (i = 1, 2, .., n)$ is the i^{th} real or complex root of $f(z)$.

The following section will discuss the interpretation of the LSTM-RNN model for obtaining the approximate roots w_i for the n^{th} order polynomial $f(z) = 0$ with real coefficients.

B. LSTM-RNN NETWORK

1) THE FUNDAMENTAL CONCEPTS BEHIND LSTM-RNN

In many applications, deep learning has received significant attention because it performs well in comparison with other NN techniques. The reason is that it can avoid gradient vanishing problem in the deep network. In fact, RNN finds a similar problem that it recognizes conditions only in a

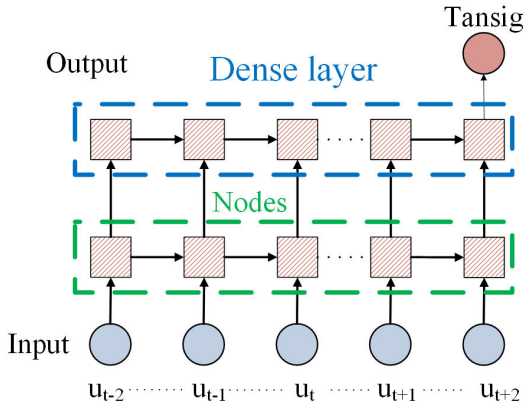

FIGURE 1. Block diagram of the LSTM model with u_t as the current and u_{t-1} the previous observations.

relatively short period, i.e., if we need the data after a short period, it may be reproducible; but once a lot of suppositions are weighted, somewhere the data get lost. A popular way for solving such issues is to use a particular sort of RNN, i.e., the LSTM-RNN [14].

Over several time steps, the LSTM retains a significant gradient. This means that the extended sequences can be used to train the network. In RNNs, an LSTM unit is made up of four major components: a memory cell and three logistic gates. The memory cell is in charge of storing data. The data flow inside the LSTM network is defined by the write, read, and forget gates. Similarly, the write gate manages writing data into the memory cell, whereas the read gate controls reading data from the memory cell and returning it to the recurrent network. The forget gate decides whether to keep or erase data from the information cell, or in the other words, how much old data to forget.

In short, these gates are the LSTM operations that perform some function on a linear combination of the network's inputs, hidden state, and prior output. In addition, LSTM is observed to be more efficient in sequence prediction than other deep learning NN. Hence, the objective of the proposed network is to interpret the roots which will factorize the polynomial into more sub-factors and then use these factors for concurrent execution in the hidden layer of the network. The key element of the LSTM-RNN is the state of cells; the state is identical to a conveyor belt with only a few insignificant linear interactions. It runs straight down to the entire hub and is very convenient for the data to move relatively and produce an active effect on response across it. The LSTM-RNN can eliminate or add information, strictly regulated by its specific gates, which are an alternative way to maintain the data respectively.

2) LSTM-RNN MODEL STRUCTURE

For a n^{th} order polynomial, a multi-layered LSTM-RNN containing hidden layers is designed to approximate the roots of the given polynomial. Similar to RNN, the hyper-parameters of the LSTM-RNN model are fitted by back propagation through time (BPTT). Hyper-parameters for our network model include the right number of layers in which training, evaluating and learning rate are the most essential and obvious ones. We design our model structure with multiple hidden layers through a network. The first hidden layer is the LSTM-RNN layer with 200 neurons and the second layer is the fully connected dense layer with 100 nodes. The block diagram of the LSTM-RNN structure is shown in Fig.1. Besides, the hyperbolic tangent sigmoid (tansig) is used as an activation function for learning parameters [23], which also calculates faster and is less prone to saturation ∼0 gradients for the network.

The input layer is fed by a vector of coefficients of a polynomial and the output layer gives the predicted roots of a given input polynomial. Additionally, limitation in the proposed model is to meet the computational efficiency with dropout rates between many layers. Although, to avoid the model overfitting factor which is tackled by increasing the LSTM layers. Furthermore, supplementary hidden layers are added during simulations while keeping certain learning parameters unchanged.

A test dataset can be used in a confirmatory way to verify that a given set of input to a given function produces some expected results. The training over multiple epochs completely passes through the datasets for evaluating the test datasets performance at each epoch to determine when to stop.

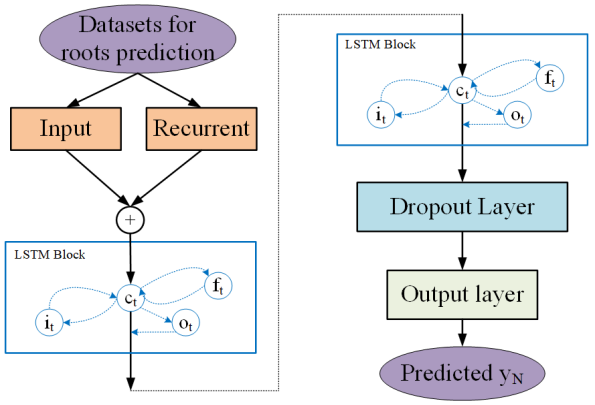

FIGURE 2. Flow chart of LSTM-RNN.

Therefore, in general, the LSTM stacking layers can improve the efficiency and estimation of the network model. Furthermore, an ADAM [24]–[26] optimizer based on RMSProp [27] and momentum as an update controller is used to improve the learning process. Finally, to normalize the results with better prediction and performance, mean square error (MSE) and mean absolute error (MAE) [28], [29] are used with an identity regression value of 0.001 and a decay rate of 1×10^{-8}.

In mathematical terminology, the output of the i^{th} hidden neuron in the network is compiled as $\hat{u}_i = z - w_i.1 = z - w_i$ where $w_i(i = 1, 2, \ldots.., n)$, the network weights of the function-to-hidden layer, i.e., the roots of the

polynomial, are to be predicted. The output of the LSTM-RNN performing multiplication with the hidden layer can be characterized as:

$$\hat{y}(z) = \prod_{i=1}^{n} \hat{\mu}_i = \prod_{i=1}^{n} (z - w_i) \tag{3}$$

The output $\hat{y}(z)$ of the network is the externally supervised learning which allows to collect data or produce a data output from the previous experience signals. In addition, if (3) is rendered after an absolute operation, the following logarithmic transformation can be extracted:

$$\bar{y} = \ln |\bar{y}(z)| \ = \sum_{i=1}^{n} \ln |(z - w_i)| \tag{4}$$

whereas, 'z' belongs to the given output polynomial.

3) LSTM-RNN ERROR COST FUNCTION

The proposed network model efficiency will be validated by a lower error cost function (ECF) value, which signifies appropriate experimental data prediction. The ECF is therefore executed as a parameter in the training phase to measure the efficiency of the LSTM-RNN network architecture to further calculate the network output for higher degree polynomials. Thus, the ECF can be defined as:

$$E(w_i) = \frac{1}{2N} \sum_{t=1}^{N} e_N^2 \ (w_i) \tag{5}$$

$$E(w_i) = \frac{1}{2N} \sum_{t=1}^{N} (u_{t+i} - y_N)^2 \tag{6}$$

Similarly, $w_i (i = 1, 2, \ldots .., n)$ is the set of all the neural network weights as for the resurrection of coefficients with polynomial roots, which is homologous to the network factorization model. Correspondingly, y_N is the predicted output value, u_{t+1} is the actual targeted value, N is the training pattern respectively. In addition, the hidden neurons compute linear divergences and transform into nonlinear hidden neurons with a logarithmic activation function, while the output neuron computes linear summation rather than multiplication. This logarithmically triggered design is compared to the $\sum - \prod$ architecture described in [8], [9]. Therefore, we need to evaluate the $y_N = \sum_{i=1}^{n} \ln |z_N - w_i|$ as a target signal and $u_{t+i} = \ln |f \ (z_N)|$ as a network actual output in the LSTM-RNN model, which can be defined as:

$$\underset{actualvector}{u_{t+i}} = \underset{network \ model}{f(u_t, u_{t-1}, \ldots u_{t-n+1})} + \underset{irreducible \ error}{\varepsilon} \tag{7}$$

where $(u_1, u_2, u_3 \ldots u_{t+i})$ with $i = 1, 2, \ldots, n$ is a series of the actual vector information in the model. Also, ε shows an irreducible error such that the data from the previous output n-time steps are used to calculate the next predicted output vector y_N. In addition, the actual vector $(u_1, u_2, u_3 \ldots u_{t+i})$ represents an input matrix $[(t - n + 1)^* n]$ and output vector $[(t - n + 1)^* 1]$ given below:

$$\begin{bmatrix} \begin{pmatrix} u_t & u_{t-1} \ldots & u_{t-n+1} \\ u_{t-1} & u_{t-2} \ldots & u_{t-n} \\ \vdots & \vdots & \vdots \\ u_n & u_{n-1} \ldots & u_1 \end{pmatrix} \end{bmatrix} \tag{8}$$

where

$$y_N = \begin{bmatrix} \begin{pmatrix} u_{t+i} \\ u_{t+i-1} \\ \vdots \\ u_{n+i} \end{pmatrix} \end{bmatrix} \tag{9}$$

Moreover, without loss of generality, the ECF objective function $L(u_t, \theta)$ can be defined as:

$$ECF = L \ (u_t, \theta) \ = \min \sum_{t}^{i} \frac{1}{2} \ (u_{t+i} - y_N)^2 \tag{10}$$

where θ depends on all the weights across the input and hidden layers with biases ignored, and is defined as $\theta = \{w_{xo}, w_{xi}, w_{xf}, w_{xc}, w_{uo}, w_{ui}, w_{uf}, w_{uc}\}$. In order to get an improved understanding of the analysis, (10) can be re-written as

$$L \ (t) = \frac{1}{2} \ (u_{t+i} - y_N)^2 \tag{11}$$

4) LSTM-RNN WITH ADAM GRADIENT EVALUATION

In this section, detailed information on how to drive gradient through ADAM optimizer in the LSTM-RNN model is discussed. ADAM is an algorithm for gradient based first-order optimization of probabilistic objective functions based on lower-order adaptive estimation. The approach is intuitive to implement, computationally efficient, with low memory needs, invariant to diagonal gradient resizing and problems that are large in terms of data and/or parameters. Hence, ADAM is based on both Momentum and RMSProp's infinity heuristics. A general implementation of ADAM optimizer is mentioned in [24]–[27]. ADAM optimizer in the LSTM-RNN model is embedded with a connection between weights and polynomial coefficients which involves a combination of two gradient descent methodologies. Besides, the momentum parameter operates to accelerate the gradient descent algorithm by taking exponentially weighted average of the gradients to train the polynomial roots. As shown in Fig. 3, the LSTM-RNN depends on the memory cell c_t has the same inputs (u_{t+1} and u_t) and outputs y_N, and have more gating units to control the flow of information. Therefore, at the time step t, we can take a partial derivative w.r.t c_t for updating the gradient as:

$$\frac{\partial L \ (t)}{\partial c_t} = \frac{\partial L \ (t)}{\partial u_t} \frac{\partial u_t}{\partial c_t} \tag{12}$$

Similarly, at time step $t - 1$, the derivative $L(t - 1)$ w.r.t. c_{t-1} can be written as:

$$\frac{\partial L \ (t-1)}{\partial c_{t-1}} = \frac{\partial L \ (t-1)}{\partial u_{t-1}} \frac{\partial u_{t-1}}{\partial c_{t-1}} \tag{13}$$

Fig. 3, shown below, describes the unfolds memory unit of the LSTM to make it easy to understand, because the LSTM-RNN model is fitted by BPTT. However, according to Fig. 3, the error is not only backpropagated via $L(t-1)$ but also from c_t. Thus, the final gradient w.r.t. c_{t-1} is defined as [30]

$$\frac{\partial L\,(t-1)}{\partial c_{t-1}} = \frac{\partial L\,(t-1)}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}} + \frac{\partial L\,(t)}{\partial u_t}\frac{\partial u_t}{\partial c_t}\frac{\partial c_t}{\partial c_{t-1}} \tag{14}$$

From c_{t-1} to c_t only elementwise multiplication by function f_t, then by chain rule, (15) can be written as:

$$dc_{t-1} = dc_t + f_t \circ dc_t \tag{15}$$

In a similar manner, (15) can be derived at any time step.

C. EVALUATION CRITERION

According to empirical results, the ADAM optimizer performs proficiently as compared to other stochastic optimization methods [31]. The well-known Mean Squared Error (MSE) and Mean Absolute Error (MAE) functions are applied to evaluate the model error that analyzes the theoretical convergence properties of the network [28], [29]. If the change of the function value becomes extremely small, it does not contribute to the learning process. As a consequence, the convergence rate has a regret constraint comparable to the best-known results in the convex optimization domain. Finally, to evaluate the LSTM-RNN model efficiency, the following MSE and MAE functions are as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_N - \hat{y_N}\right)^2 \tag{16}$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|\left(y_N - \hat{y_N}\right)\right| \tag{17}$$

where y_N denotes the actual i^{th} root ($i = 1, 2, \ldots, n$) in the test dataset, and $\hat{y}_N$ is the corresponding approximation obtained with the proposed LSTM-RNN approach.

D. DATA SET NORMALIZATION WITH GENERATION

Normalization is the process of restructuring data into a network that satisfies two basic requirements: (1) there is no data redundancy; and (2) data dependencies are logical (all related data objects are stored together). Therefore, in this study, datasets are scaled in the range $[-1, 1]$ to improve the training algorithm convergence characteristics. Using MATLAB, we generate datasets of polynomial coefficients against the corresponding degree n by randomly choosing (uniformly distributed random numbers) with 10,000 examples in each degree. Hence, for $n = 5$, 10, 15 and 20 degree polynomials, the generated datasets are 50000, 100000, 150000 and 200000 respectively. Meanwhile, from the coefficient datasets, the exact (real or complex) roots are calculated using a symbolic computation package. Thus, the LSTM-RNN does not know a priori which roots are real or complex. It is important to note here that double-precision values are used to generate these datasets although coefficients and roots are taken with only four decimal places. Hence, the coefficient datasets of a particular polynomial degree n are used as an input with the polynomial roots as an output, which is then processed by the LSTM-RNN model to produce an approximation of the roots of a real n degree polynomial.

Tables 1 and 2, respectively show the head of the datasets that are used with the LSTM-RNN to train and compute approximations of the roots (real and complex) for $n = 5$ as an example. The real and complex parts of the corresponding roots are represented in Table 2 by the odd and even columns, respectively, i.e.:

$$w_i = \{\text{Re}(w_i), \text{Im}(w_i)\}, \quad i = 1, \ldots, n \tag{18}$$

Furthermore, in this study, 80% of our datasets are for the training set, while the 20% remaining datasets are to test and validate the model. The head of datasets for polynomial of degrees 10, 15 and 20 is not tabulated due to large number of data values and hence only the simulation results are shown.

TABLE 1. Head of the input datasets for training the LSTM-RNN ($n = 5$) for computing roots.

a_0	a_1	a_2	a_3	a_4	a_5
0.0535	0.8338	-0.9358	1.4290	0.1071	0.8338
0.0386	-0.0602	0.1356	-0.5240	-0.6095	-0.0039
-0.3726	-0.1238	0.0383	0.5078	-0.6944	0.6065
0.8338	-0.6869	-0.8255	0.6451	-0.2459	0.5816
-0.5497	-0.2201	-0.2541	-0.3062	-0.5404	0.4292
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.

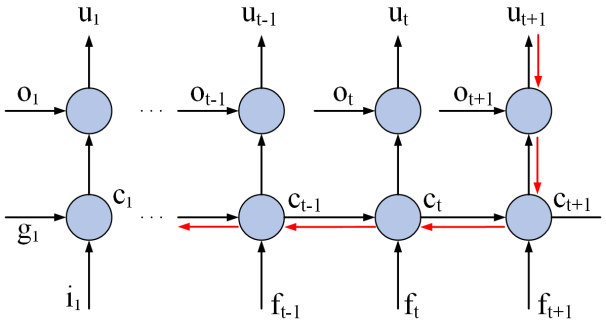

FIGURE 3. Unfolds memory unit of LSTM.

III. RESULTS AND DISCUSSION

In this section, the obtained datasets and approximate roots based on the proposed network methodology are discussed. The proposed technique has been tested in the following way: the datasets are first generated using MATLAB programming. Then the confusion matrix approximations [32]

TABLE 2. Head of the output datasets for training the LSTM-RNN ($n = 5$) for computing roots.

$Re(w_1)$	$Im(w_1)$	$Re(w_2)$	$Im(w_2)$	$Re(w_3)$	$Im(w_3)$	$Re(w_4)$	$Im(w_4)$	$Re(w_5)$	$Im(w_5)$
-16.700	0.0000	0.8068	0.0000	0.8608	-1.0961	0.5147	0.6686	-0.2363	0.6686
2.7833	-0.4371	-0.1809	1.2771	-0.4371	-2.5625	-0.8543	0.0000	-0.0065	0.0000
-1.2294	1.0961	0.0000	1.2771	-0.1809	-2.5625	1.1262	0.0976	0.0000	0.6451
1.0916	0.6907	1.0916	-0.3039	-1.1642	0.0000	1.0062	0.6761	0.0000	-0.6761
-16.700	0.3039	-0.8884	-0.6907	0.4308	0.0976	0.4308	-1.0062	-0.2363	0.0000
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.	.

TABLE 3. Estimated datasets generation accuracy for different polynomial degrees.

Highest Polynomial Degree	*Classifier*	*Prediction speed obs/sec*	*Execution time (sec)*	*Number of learners*	*Observation*	*Accuracy (%)*
5th	Bagged tree	18000	232.6	30	50000	99.4%
10th	Bagged Tree	29000	311.2	30	100000	99.3%
15th	Bagged Tree	35000	739.2	30	150000	98.8%
20th	Bagged Tree	42000	1021.3	30	200000	98.2%

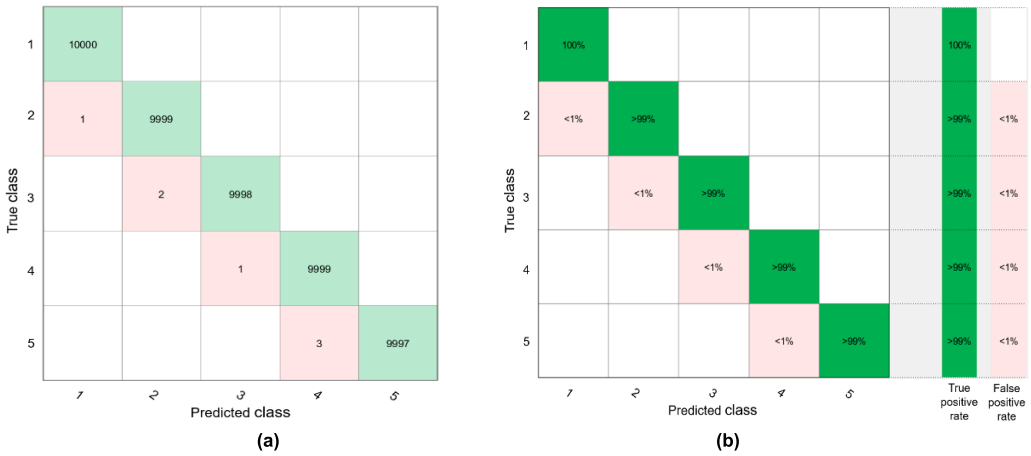

FIGURE 4. (a) Generated datasets validation using confusion matrix for $n = 5$ (b) Percentage datasets accuracy.

remain used to verify the effectiveness of the generated datasets. Finally, the LSTM-RNN model is tested and validated using python tool for computing roots of a given polynomial degree n. In this study, simulations have been performed using Intel core i-7 with a CPU clock of 1.8Ghz and 8 Gb RAM for $f(z)$ with $n = 5$, 10, 15 and 20 respectively.

A. MATLAB SIMULATION ANALYSIS

Figure 4 shows the confusion matrix based on the datasets for $n = 5$. The confusion matrix is developed using the MATLAB classifier learner. The matrix can help us identify the areas where the classifier model has performed poorly. Fig. 4(a) shows the percentage datasets accuracy for the polynomial coefficients for $n = 5$ only. However, the observations are similar for other degrees.

The rows represent the true class, and the columns show the predicted class with the diagonal percentage values displaying the best approximation and accuracy of the datasets as shown in Fig. 4(b). In this case, the percentage accuracy for the polynomial with $n = 5$ is found to be 99.4%. Similarly, Fig. 5 shows the analysis for $n = 10$ with the percentage accuracy being 99.2%. Correspondingly, the percentage accuracy for polynomial of degrees 10 and 20 are also found to be over 98%. Therefore, the results justify that the generated datasets for different cases of polynomial degrees are effective and valid.

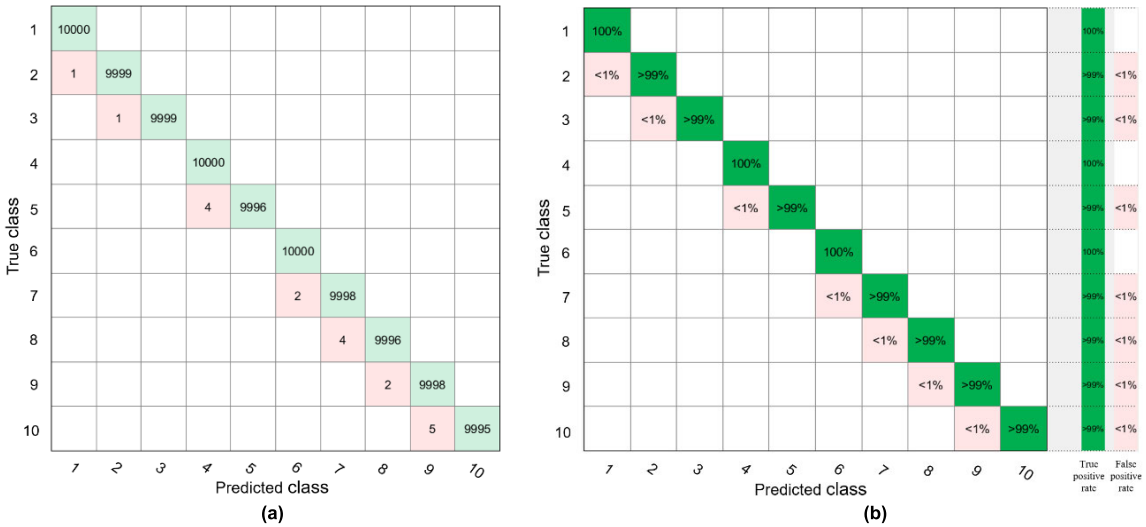

FIGURE 5. (a) Generated datasets validation using confusion matrix for $n = 10$ (b) Percentage datasets accuracy.

TABLE 4. Simulation results for different polynomial degrees with different epochs.

Polynomial Degree	*Epoch*	*Mean Square Error*	*Mean Absolute Error*	*% Accuracy*	*Validation Loss*	*Validation Accuracy*	*Validation Mean Square Error*	*Validation Mean Absolute Error*
5	2000	5.46×10^{-04}	1.10×10^{-04}	99.82%	8.40×10^{-04}	99.84%	3.83×10^{-04}	7.99×10^{-04}
5	4000	3.36×10^{-04}	6.39×10^{-04}	99.88%	6.90×10^{-04}	99.87%	3.80×10^{-04}	5.66×10^{-04}
5	6000	2.45×10^{-04}	4.29×10^{-04}	99.89%	5.40×10^{-04}	99.95%	1.19×10^{-04}	1.94×10^{-04}
10	2000	2.93×10^{-03}	9.19×10^{-03}	92.25%	15.3×10^{-03}	92.78%	5.79×10^{-03}	7.70×10^{-03}
10	4000	1.85×10^{-03}	5.85×10^{-03}	94.83%	11.5×10^{-03}	94.87%	4.94×10^{-03}	5.79×10^{-03}
10	6000	1.02×10^{-03}	2.05×10^{-03}	97.75%	8.93×10^{-03}	97.67%	1.76×10^{-03}	1.72×10^{-03}
15	6000	1.66×10^{-03}	4.81×10^{-03}	95.35%	9.80×10^{-03}	95.78%	2.77×10^{-03}	2.77×10^{-03}
20	6000	2.43×10^{-03}	6.93×10^{-03}	93.63%	11.8×10^{-03}	94.19%	5.11×10^{-03}	6.30×10^{-03}

Table 3 shows the summarized results with classifier learning settings for predicting the datasets generation accuracy for different cases of polynomial degree n.

B. LSTM-RNN MODEL VERIFICATION

In order to compute the polynomial roots based on the datasets, python programming with 3.7 documentation series is used to implement the proposed LSTM-RNN model with ADAM optimizer. In fact, validating the model is also necessary to rely on the model based on its evaluation through the valid datasets. Hence, it is essential to evaluate the model on the validation dataset before testing on a training dataset. For achieving this task there are two ways: (1) Taking the validation dataset from the training dataset; and (2) Keeping different validation sets while splitting the main datasets. Such approaches are being used by many algorithms, including the famous Random Forest algorithm [33]. Hence, in our network model the head of datasets consists of both inputs and desired (or target) outputs data. Furthermore, to update the weights of the LSTM model, the validation dataset has only input values where verification of the loss and accuracy measures on the training set, while val_loss and val_acc are measures on the validation set. The following information shows the parameters setting of the LSTM-RNN model during execution period for polynomials of degrees 5, 10, 15 and 20. Other settings for the LSTM-RNN model are n_test = 100, n_val = 100, draw = 1, n_nodes_hl (hidden layer) = [80], n_nodes_dense = [100], n_nodes_LSTM = [200, 200], n_drop_out = [0.2,0.1, 0.2] and verbose = [1]. Similarly, the identity regression of 0.1 and exponentially decaying rate of $e^{*}10^{-6}$ with validation split of 0.1 are the design parameters set in the network model. Figs. 6(a)-(c) show the polynomial case of $n = 5$ under different epochs. The polynomial roots approximation is evaluated in terms of both training accuracy and training loss w.r.t validation accuracy. The results show that the polynomial roots are computed efficiently without any overfitting. Also, the LSTM-RNN model with 2000 epochs has an accuracy over 99.82% with validation loss of 0.08%. Similarly, using both 4000 and 6000 epochs, the training model and validation accuracy is above 99% with validation loss less than 0.5% respectively.

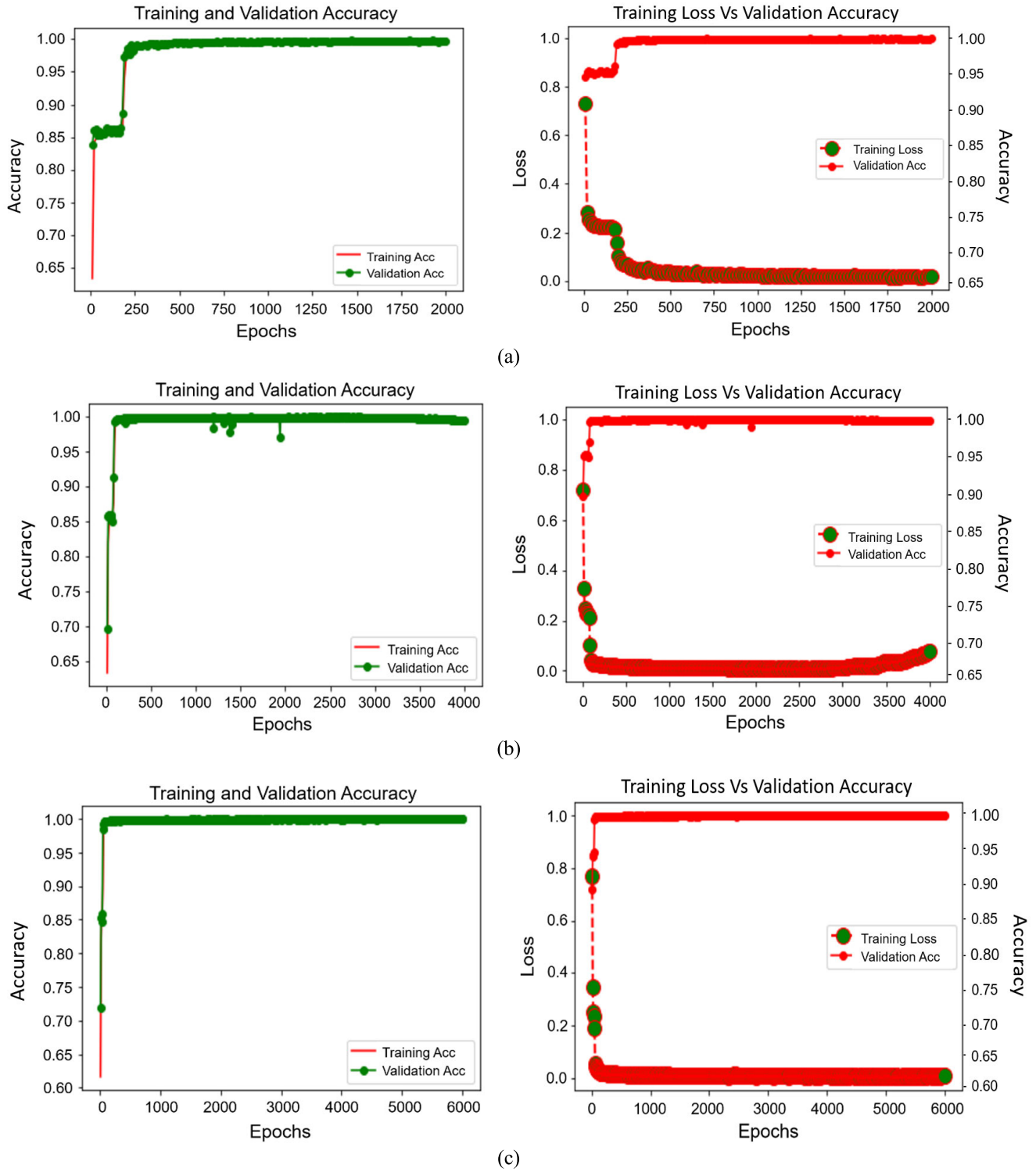

FIGURE 6. Validation accuracy vs training loss and training accuracy results for 5th-degree polynomials with (a) 2000 epochs (b) 4000 epochs (c) 6000 epochs.

Similarly, Figs. 7(a)-(c) show the case for $n = 10$ polynomial degree using the aforementioned network model parameters setting. In this case the percentage accuracy using 2000, 4000 and 6000 epochs are respectively 92.7%, 94.8% and 97.6%.

Figs. 8 and 9 show the cases for polynomials of degrees 15 and 20, respectively. From the aforementioned analysis it is observed that as the number of epochs increases the validation accuracy will also increase. Therefore, for $n = 15$ and 20, the analysis is only performed for the highest epochs case

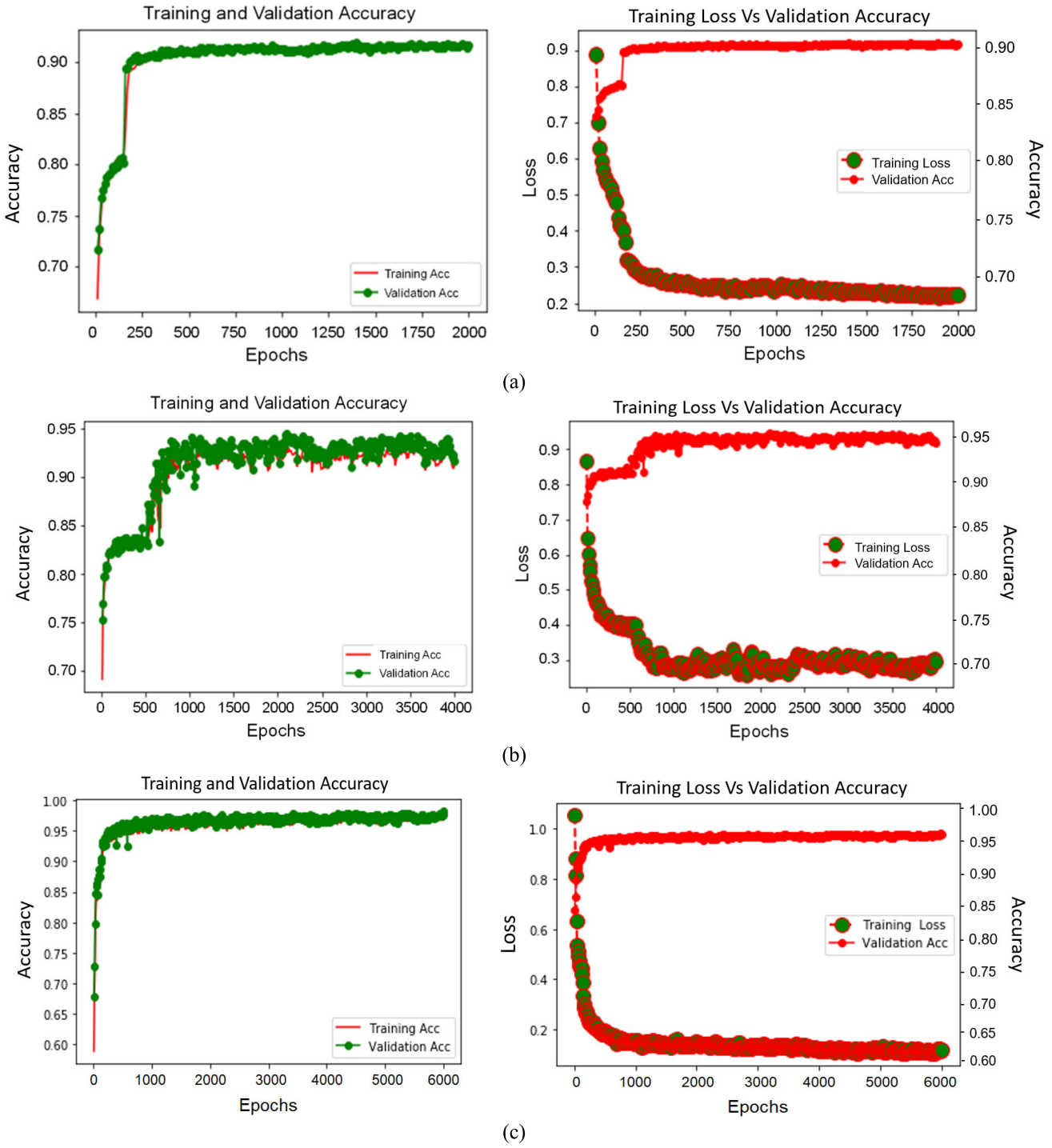

FIGURE 7. Validation accuracy vs training loss and training accuracy results for 10th-degree polynomials with (a) 2000 epochs (b) 4000 epochs (c) 6000 epochs.

i.e., 6000 epochs. The validation accuracies for $n = 15$ and 20 are 95.2% and 93.6% respectively. Hence, the proposed methodology for roots prediction can be up to 20th degree. Moreover, it is obvious that the validation accuracy for higher degrees can be further improved by increasing the number of epochs, with however up surging execution time and system resources.

Table 4 shows the summarized results under different cases of polynomial degrees with different epochs.

Table 5 shows the comparative simulation analysis of roots approximation for polynomials of degrees 5, 10, 15 and 20. In this study, the comparative analysis is performed with the conventional FNN based ANN model. As mentioned in section I, the most commonly used ANN tech-

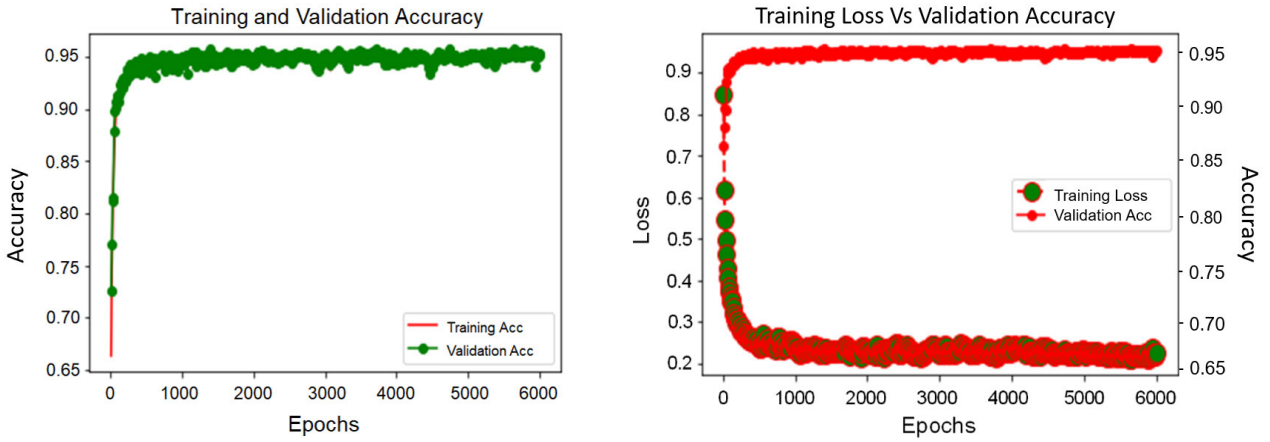

FIGURE 8. Validation accuracy vs training loss and training accuracy results for 15th-degree polynomials with 6000 epochs.

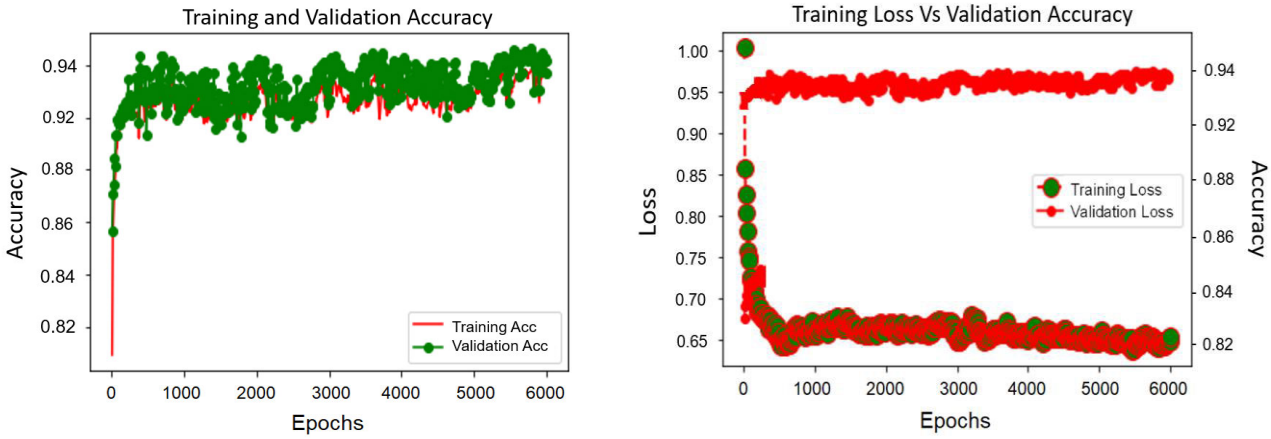

FIGURE 9. Validation accuracy vs training loss and training accuracy results for 20th-degree polynomials with 6000 epochs.

TABLE 5. Simulation results for comparative analysis between conventional FNN-ANN and proposed LSTM-RNN.

Methodology	*Degree*	*Mean Square Error (MSE)*	*Exec time (s)*	*Epochs*	*% Accuracy*
Conventional FNN-ANN	5	5.48×10^{-03}	4.67	6000	96.84%
	10	2.47×10^{-02}	7.52	6000	94.67%
	15	4.17×10^{-02}	11.4	6000	91.43%
	20	7.58×10^{-02}	14.3	6000	89.33%
Proposed LSTM-RNN	5	2.45×10^{-04}	12.4	6000	99.82%
	10	1.02×10^{-03}	17.5	6000	97.75%
	15	1.66×10^{-03}	23.7	6000	95.35%
	20	2.43×10^{-03}	26.1	6000	93.63%

nique for polynomial roots approximation in the previous research work is the FNN-ANN technique. Therefore, the same ANN technique is employed for comparison with the proposed methodology. The performance indices for comparison are the mean square error (MSE), execution time and % accuracy under the fixed epochs i.e., 6000. From Table 4, the comparative results clearly demonstrate that the proposed LSTM-RNN model surpasses the FNN-ANN model for roots approximation in terms of the accuracy and lower MSE at the cost of slightly higher execution time. In fact, as stated in section II, due to the memory cells in the LSTM-RNN model, the execution time is obviously more than the FNN based ANN model. Therefore, there is a trade-off between the accuracy and the execution time. However,

the execution time could be reduced with better system resources.

The graphical roots prediction accuracies under different cases of polynomial degrees are shown in Fig. 10. The results conclude that using the proposed LSTM-RNN model, the roots prediction percentage accuracies for polynomials of degrees 5, 10, 15 and 20 evaluated at 6000 epochs are 99.8%, 97.7%, 95.3% and 93.6%, respectively; whereas with the FNN-ANN model, the percentage accuracies are 96.8%, 94.6%, 91.4% and 89.3%.

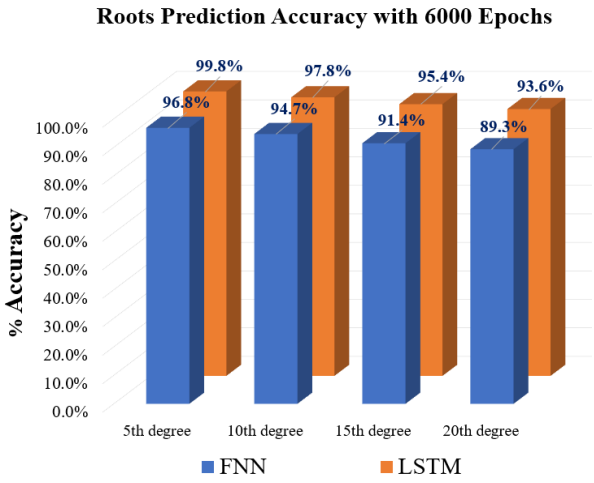

FIGURE 10. Model prediction accuracy for different polynomial degrees.

IV. CONCLUSION

This paper presents a performance evaluation of the LSTM recurrent neural network (RNN) for approximating roots of a given polynomial. The study started by generating datasets as an input and target data for testing the LSTM-RNN model for different cases of polynomial degrees. The proposed model weights were then modified based on the difference between the actual root values and the values generated by the model, i.e., approximations to the polynomial roots. Finally, several experimental results were performed and the consequences were evaluated in terms of efficiency and validation accuracy for different cases of polynomial degrees. Moreover, the proposed LSTM-RNN model was compared with the commonly used NN model (namely FNN based ANN model) for polynomial roots approximation. Simulation results clearly demonstrated that the proposed LSTM-RNN model outperforms the FNN-ANN model in terms of accuracy and MSE at the cost of slightly higher execution time.

From the results, although the percentage efficiency for a higher degree polynomial is slightly lower but the performance can be improved by increasing the number of epochs. Therefore, it can be inferred that the proposed LSTM-RNN techniques is efficient and feasible in computing any real univariate polynomial with real and complex roots.

In summary, the novelties of the present study are:

1. A more advanced DNN technique, i.e., the LSTM-RNN has been used for the first time in the proposed methodology for approximating the polynomials roots.
2. The validity of the coefficients and roots datasets have also been evaluated successfully using the MATLAB confusion matrix.
3. Finally, several experimental have been performed to prove the validity of the proposed methodology. Furthermore, the results are compared and examined with the conventional FNN based ANN. The results clearly demonstrate the superiority of the proposed LSTM-RNN model for approximating the roots of the polynomials.

Though the results presented in this paper are preliminary for a particular class of polynomials, namely polynomials with real coefficients, they are highly promising and indicate the potential of the LSTM-RNN based NN model for computing the polynomial roots. In further research works, we will consider the case of finding the roots for multivariate polynomials.

ACKNOWLEDGMENT

The authors would like to thanks Shenzhen Scientific and Technological Fund for technical and DSR for financial support.

REFERENCES

[1] B. Kalantari, *Polynomial Root-Finding and Polynomiography*. Singapore: World Scientific, 2008.
[2] J. M. McNamee and P. Victor, *Numerical Methods for Roots of Polynomials–Part II*. Amsterdam, The Netherlands: Elsevier, 2013.
[3] D. S. Huang, "A neural network based factorization model for polynomials in several elements," in *Proc. 5th Int. Conf. Signal Process. 16th World Comput. Congr.*, Beijing, China, Aug. 2000, pp. 1617–1622.
[4] D. S. Huang, "Application of neural networks to finding real roots of polynomials in one element," in *Proc. ICONIP*, Taejon, South Korea, vol. 2, Nov. 2000, pp. 1108–1113.
[5] D. R. Hush and B. G. Horne, "Progress in supervised neural networks," *IEEE Signal Process. Mag.*, vol. 10, no. 1, pp. 8–39, Jan. 1993.
[6] D.-S. Huang and Z. Chi, "Neural networks with problem decomposition for finding real roots of polynomials," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Washington, DC, USA, Jul. 2001, p. A25.
[7] S. J. Perantonis, N. Ampazis, S. J. Varoufakis, and G. Antoniou, "Factorization of 2-D polynomials using neural networks and constrained learning techniques," in *Proc. IEEE Int. Symp. Ind. Electron.*, vol. 3, Jul. 1997, pp. 1276–1280.
[8] R. Hormis, G. Antonion, and S. Mentzelopoulou, "Separation of two-dimensional polynomials via a neural net," in *Proc. Int. Conf. Modeling Simulation*, Pittsburg, PA, USA, 1995, pp. 304–306.
[9] D. S. Huang, "Finding roots of polynomials based on root moments," in *Proc. 8th Int. Conf. Neural Inf. Process. (ICONIP)*, vol. 3, 2001, pp. 1565–1571.
[10] D.-S. Huang, "A constructive approach for finding arbitrary roots of polynomials by neural networks," *IEEE Trans. Neural Netw.*, vol. 15, no. 2, pp. 477–491, Mar. 2004.
[11] D. Freitas, L. G. Lopes, and F. Morgado-Dias, "A neural network-based approach for approximating arbitrary roots of polynomials," *Mathematics*, vol. 9, no. 4, p. 317, 2021.
[12] S. Vani, T. V. Madhusudhana Rao, and C. Kannam Naidu, "Comparative analysis on variants of neural networks: An experimental study," in *Proc. 5th Int. Conf. Adv. Comput. Commun. Syst. (ICACCS)*, Mar. 2019, pp. 429–434.
[13] T. Zheng, Z. Chen, S. Ding, and J. Luo, "Enhancing RF sensing with deep learning: A layered approach," *IEEE Commun. Mag.*, vol. 59, no. 2, pp. 70–76, Feb. 2021.
[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] B. B. Sahoo, R. Jha, A. Singh, and D. Kumar, "Long short-term memory (LSTM) recurrent neural network for low-flow hydrological time series forecasting," *Acta Geophys.*, vol. 67, no. 5, pp. 1471–1481, 2019.
[16] M. Jiao and D. Wang, "The Savitzky-Golay filter based bidirectional long short-term memory network for SOC estimation," *Int. J. Energy Res.*, vol. 45, no. 13, pp. 19467–19480, 2021.
[17] J. Ma, H. Liu, C. Peng, and T. Qiu, "Unauthorized broadcasting identification: A deep LSTM recurrent learning approach," *IEEE Trans. Instrum. Meas.*, vol. 69, no. 9, pp. 5981–5983, Sep. 2020.
[18] A. Akbari Asanjan, T. Yang, K. Hsu, S. Sorooshian, J. Lin, and Q. Peng, "Short-term precipitation forecast based on the PERSIANN system and LSTM recurrent neural networks," *J. Geophys. Res., Atmos.*, vol. 123, no. 22, pp. 12543–12563, 2018.
[19] D. Chen, J. Zhang, and S. Jiang, "Forecasting the short-term metro ridership with seasonal and trend decomposition using loess and LSTM neural networks," *IEEE Access*, vol. 8, pp. 91181–91187, 2020.
[20] S. Aladin, A. V. S. Tran, S. Allogba, and C. Tremblay, "Quality of transmission estimation and short-term performance forecast of lightpaths," *J. Lightw. Technol.*, vol. 38, no. 10, pp. 2807–2814, May 15, 2020.
[21] D. Huang, "A neural root finder of polynomials based on root moments," *Neural Comput.*, vol. 16, no. 8, pp. 1721–1762, Aug. 2004.
[22] I. E. Livieris, "Improving the classification efficiency of an ANN utilizing a new training methodology," *Informatics*, vol. 6, no. 1, p. 1, 2019.
[23] A. Sengupta, A. Seal, C. Panigrahy, O. Krejcar, and A. Yazidi, "Edge information based image fusion metrics using fractional order differentiation and sigmoidal functions," *IEEE Access*, vol. 8, pp. 88385–88398, 2020.
[24] R. Wang, W. Wang, T. Ma, and B. Zhang, "An adaptive gradient method with differentiation element in deep neural networks," in *Proc. 15th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Nov. 2020, pp. 1582–1587.
[25] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.
[26] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Represent. (ICLR)*, 2015, pp. 1–15.
[27] F. Zou, L. Shen, Z. Jie, W. Zhang, and W. Liu, "A sufficient condition for convergences of Adam and RMSProp," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, no. 1, Jun. 2019, pp. 11119–11127.
[28] W. Wang and Y. Lu, "Analysis of the mean absolute error (MAE) and the root mean square error (RMSE) in assessing rounding model," in *Proc. IOP Conf., Mater. Sci. Eng.*, 2018, vol. 324, no. 1, Art. no. 012049.
[29] C. J. Willmott and K. Matsuura, "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance," *Climate Res.*, vol. 30, no. 1, pp. 79–82, 2005.
[30] G. Chen, "A gentle tutorial of recurrent neural network with error backpropagation," 2016, *arXiv:1610.02583*.
[31] K. Marti, "Stochastic optimization methods," in *Stochastic optimization methods*. Berlin, Germany: Springer, 2015.
[32] T. C. W. Landgrebe and R. P. W. Duin, "Efficient multiclass ROC approximation by decomposition via confusion matrix perturbation analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 810–822, May 2008.
[33] M. Schonlau and R. Y. Zou, "The random forest algorithm for statistical learning," *Stata J.*, vol. 20, no. 1, pp. 3–29, Mar. 2020.

MADIHA BUKHSH was born in Rawalpindi, Pakistan, in 1989. She received the Bachelor of Science degree from the University of the Punjab, Lahore, Pakistan, in 2011, and the master's degree in mathematics from Preston University, Islamabad, in 2016. She is currently pursuing the Ph.D. degree with the Department of Engineering Mechanics, School of Aeronautics and Astronautics, Zhejiang University, China. Her current research interests include machine learning, deep learning, and application of machine learning in solid and fluid mechanics.

MUHAMMAD SAQIB ALI was born in Rawalpindi, Pakistan, in 1981. He received the B.S. degree in electronics engineering from Hamdard University, Karachi, Pakistan, in 2004, the M.S. degree in electrical engineering and computer science from Seoul National University, South Korea, in 2010, and the Ph.D. degree in electrical engineering from Zhejiang University, China, in 2021. Since 2004, he has been with the Faculty of the Electrical and Computer Engineering, COMSATS University Islamabad (CUI), Pakistan, where he was a Lecturer, from 2007 to 2011. He has been also working as an Assistant Professor, since 2011. His current research interests include AC/DC rectifiers systems, DC-DC converters, application of power electronics in renewable energies, wide band gap devices, parameters optimization using artificial intelligence techniques, space power systems, and photovoltaic power conditioning systems.

MUHAMMAD USMAN ASHRAF received the Ph.D. degree in computer science from King Abdulaziz University, Saudi Arabia, in 2018. He has worked as a HPC Scientist with the HPC Centre, King Abdulaziz University. He is currently an Assistant Professor and the Head of the Department of Computer Science, Government College Women University, Sialkot, Pakistan. His research on exascale computing systems, high performance computing (HPC) systems, parallel computing, HPC for deep learning, location-based services system has appeared in IEEE Access, *IET Software*, *International Journal of Advanced Research in Computer Science*, *International Journal of Advanced Computer Science and Applications*, *International Journal of Information Technology and Computer Science*, *International Journal of Computer Science and Security*, and several international IEEE/ACM/Springer conferences.

KHALID ALSUBHI received the B.Sc. degree in computer science from King Abdulaziz University (KAU), Jeddah, Saudi Arabia, in 2003, and the Master of Mathematics (M.Math.) and Ph.D. degrees in computer science from the University of Waterloo, Waterloo, ON, Canada, in 2009 and 2016, respectively. He is currently an Associate Professor in computer science with KAU. His research interests include network security and management, cloud computing, and the security and privacy of healthcare applications.

WEIQIU CHEN (Member, IEEE) received the B.S. and Ph.D. degrees from Zhejiang University, China, in 1990 and 1996, respectively. He is currently a Professor with the Department of Engineering Mechanics, Zhejiang University. He was worked as a Postdoctoral Research Associate with The University of Tokyo, from 1997 to 1999. He was promoted to an Associate Professor, in 1999, and a Full Professor, in 2000. He has engaged himself in mechanics of soft materials and structures, mechanics of smart materials/structures, and vibration/waves in structures for over 30 year's. He has coauthored over 300 peer-reviewed journal articles and three English books on elasticity of transversely isotropic materials, piezoelectricity, and Green's functions, respectively. His H-index of 51. He is currently serving as a editorial member (or an associate editor-in-chief) of more than a dozen of academic journals, including *Mechanics of Advanced Materials and Structures*, *Journal of Thermal Stresses*, and *Applied Mathematics and Mechanics* (English Edition).