Neurocomputing 507 (2022) 292–310

Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

MFRFNN: Multi-Functional Recurrent Fuzzy Neural Network for Chaotic Time Series Prediction

Hamid Nasiri, Mohammad Mehdi Ebadzadeh *

Department of Computer Engineering, Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran

A R T I C L E   I N F O

Article history:
Received 13 January 2022
Revised 28 July 2022
Accepted 3 August 2022
Available online 8 August 2022
Communicated by Zidong Wang

Keywords:
Chaotic time series
Neuro-fuzzy inference system
Prediction
Recurrent fuzzy neural network
Time series forecasting

A B S T R A C T

Chaotic time series prediction, a challenging research topic in dynamic system modeling, has drawn great attention from researchers around the world. In recent years extensive researches have been done on developing chaotic time series prediction methods, and various models have been proposed. Among them, recurrent fuzzy neural networks (RFNNs) have shown significant potential in this area. Most of the proposed RFNNs learn a single function, but when dealing with chaotic time series, different outputs may be generated for a specific input based on the system's state. So, a network is required that can learn multiple functions simultaneously. Based on this concept, a novel multi-functional recurrent fuzzy neural network (MFRFNN) is proposed in this paper. MFRFNN consists of two fuzzy neural networks with Takagi-Sugeno-Kang fuzzy rules, one is used to produce the output, and the other to determine the system's state. There is a feedback loop between these two networks, which makes MFRFNN capable of learning and memorizing historical information of past observations. Employing the states allows the proposed network to learn multiple functions simultaneously. Moreover, a new learning algorithm, which employs the particle swarm optimization algorithm, is developed to train the networks' weights. The effectiveness of MFRFNN is validated using the Lorenz and Rossler chaotic time series and four real-world datasets, including Box–Jenkins gas furnace, wind speed prediction, Google stock price prediction, and air quality index prediction. Based on the root mean square error, the proposed method shows a decrease of $35.12\%, 13.95\%$, and 49.62% from the second best methods in the Lorenz time series, Box–Jenkins gas furnace, and wind speed prediction dataset, respectively.

1. Introduction

As a mathematical theory, chaos theory has been one of the hot research topics for decades. The idea behind it is that unpredictable and random behavior can occur in a system that follows deterministic laws, implying that a specific initial condition constantly evolves in the same way. Poincare was the first to discover a chaotic system, thus establishing the groundwork for modern chaos theory [1]. Since then, many researchers have worked on this theory and developed it rapidly. Following the development of chaos theory, this theory has been used by researchers in various fields. The chaotic time series, one of the most popular chaos applications, connects the real world and chaos theory like a bridge [2,1].

Chaotic time series are series of consecutive observations gathered from a chaotic system. For predicting the system's behavior, past data points are used to build a model that represents the underlying dynamic of that system [3,4] This is a very effective modeling method when there is not enough knowledge about the underlying data generation process or when the relationship between the observed variables and the prediction variable is not obvious [5]. In recent years extensive researches have been done on the development of chaotic time series prediction methods, and various models have been proposed, including Artificial Neural Networks (ANN) [4,6–8], Fuzzy Neural Networks (FNN) [1], Autoregressive models [9], Swarm Intelligence based models [10]. Among these models, FNNs and ANNs are the best models as they have great ability in handling nonlinearity [11–13]. It has been proved theoretically that ANNs and FNNs are universal approximators capable of estimating an arbitrary nonlinear function to any desired accuracy [14–16].

Although feed forward neural networks can map the static input–output relationship, they are unsuitable for modeling the chaotic time series [17]. Despite the poor performance of the feed forward structure in predicting time series, Recurrent Neural Network (RNN) has shown the significant potential of learning temporal dependencies within time series data [18,19]. Due to the

* Corresponding author.
E-mail addresses: h.nasiri@aut.ac.ir (H. Nasiri), ebadzadeh@aut.ac.ir (M.M. Ebadzadeh).

https://doi.org/10.1016/j.neucom.2022.08.032
0925-2312/© 2022 Elsevier B.V. All rights reserved.

existence of a feedback loop in RNN's structure, it can learn and memorize information of the past observations by forming the structure of information circulation [20]. RNNs are computationally powerful and significantly compact compared to feed-forward networks for the same approximation accuracy. Moreover, RNNs have proven to be universal approximators [21]. Although RNNs have good prediction ability, it is difficult to train them due to the vanishing or exploding gradient problem. These problems lead to slow convergence and higher computational requirements [22].

Other methods showing promising performance for time series prediction are FNNs. FNNs are hybrid methods that combine an ANN's learning capabilities with a fuzzy system's interpretability and semantic transparency. FNNs have a considerable advantage in terms of local representation and human reasoning and have proven to be quite effective in dealing with nonstochastic uncertainties [14]. Since FNNs are capable of capturing the underlying relationship from the data, they have achieved great success in time series prediction [23]. To combine RNNs potential of learning temporal dependencies with FNNs capability to deal with fuzzy information, Recurrent Fuzzy Neural Network (RFNN) has been proposed in the literature [24–30].

Although the proposed RFNNs can learn temporal dependencies and memorize historical information, most of them learn a single function, so they generate a specific output based on the current and previous inputs in each time step. But when dealing with chaotic time series (i.e., strong nonlinear problems), different outputs may be generated for a specific input based on the system's state. To better explain the problem through visualization, we illustrated the return map of the Mackey–Glass chaotic time series in Fig. 1. As it can be seen, in x^*, there are two possible outputs (i.e., α and β), and based on the system's state, the output can be different. In this problem, the algorithm needs to learn two different functions simultaneously (i.e., F_1 and F_2) using two states to let it select one based on the system's state. Therefore, if an algorithm learns a single function, it can not determine the output value in x^* and, as a result, can not achieve high accuracy in time series prediction. So, a network is required to determine the system's state and learn a single function for each state. In other words, the system should be capable of learning multiple functions simultaneously. Another issue to be addressed is the fact that chaotic time series are highly sensitive to initial conditions, leading to long-term unpredictability characteristics [5,1,31]. Therefore, a network with long-term prediction ability requires learning the states of the system to capture the dynamic behavior of the chaotic time series. It also needs a structure with a feedback loop to memorize historical information of past observations.

In light of the requirements outlined above, a novel multi-functional recurrent fuzzy neural network (MFRFNN) for time series prediction has been proposed in this paper. MFRFNN consists of two FNNs, one to predict the future value of time series and the other to determine the state of the system. The proposed network has a feedback loop between two networks to learn and memorize historical information of past observations. Furthermore, it employs the states to learn multiple functions simultaneously that result in capturing the dynamic characteristics of the chaotic time series and predicting long-term values of the time series. To the best of the authors' knowledge, MFRFNN is the first RFNN that determines the system's state, learns a single function for each state, and models multiple functions simultaneously. The major contributions of this paper are summarized as follows:

1. A novel multi-functional recurrent fuzzy neural network for time series prediction is proposed, which combines the advantages of an RNN and an FNN. MFRFNN consists of two FNNs that connect with a feedback loop. This helps MFRFNN to memorize past observations and capture the dynamic characteristics of chaotic time series. Moreover, it determines the system's state and is capable of learning multiple functions simultaneously.
2. Developing a new learning algorithm to learn weights of MFRFNN. This algorithm employs the least square method and particle swarm optimization algorithm to learn the weights of two FNNs.
3. The effectiveness of MFRFNN is evaluated with the Lorenz and Rossler chaotic time series and four real-world time series. The experimental results show that MFRFNN effectively forecasts time series, and the prediction accuracy is considerably increased. For the Lorenz time series, Box–Jenkins gas furnace dataset, and wind speed prediction dataset, based on the root mean square error, the proposed method showed a decrease of 35.12%, 13.95%, and 49.62% from the second best methods, respectively.

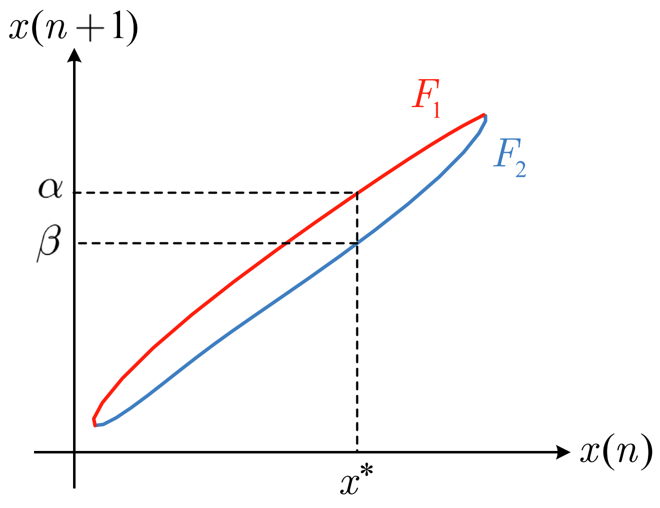


Fig. 1. Return map of Mackey–Glass chaotic time series.

The rest of the paper is organized as follows: Section 2 reviews the related works. Section 3 briefly introduces the particle swarm optimization (PSO) algorithm. The proposed method is then introduced in Section 4, with Section 5 evaluating the performance of MFRFNN and giving the experimental results. Section 6 presents a discussion of the results, and finally Section 7 concludes the paper with a short summary.

2. Related works

In recent years, time series prediction has attracted broad attention from researchers, it became more and more popular as time goes by, and various methods have been proposed for it. ANN is one of these methods. Many researchers have developed novel radial basis function neural networks (RBFNN) for time series prediction tasks. Li et al. [32] proposed ECA-Adam-RBFNN, which uses an enhanced clustering algorithm and the Adam algorithm to train an RBFNN. They used K-means with ant colony optimization to determine the center of RBFS. Also, the Adam algorithm was used to adjust the centers and weights of the network. Zhu and Meng [33] developed RBFNN-GA for gross domestic product prediction in 2021. RBFNN-GA employs the genetic algorithm to tune the parameters of the RBFNN. Han et al. [34] proposed a novel self-organizing RBFNN, which uses an accelerated second-order learning algorithm to optimize the structure and parameters of RBFNN simultaneously. Another method is RNN, showing great ability in dealing with temporal dependencies [18]. Different variants of

RNN have been proposed in the literature, such as long short-term memory (LSTM), gated recurrent unit (GRU), and echo state network (ESN).

ESN, proposed by Jaeger and Has [35], has a good performance for chaotic time series prediction. It is composed of an input layer, a reservoir of sparsely connected and randomly-generated neurons, and an output layer. It has the RNN's potential of learning temporal dependencies but does not require training of the internal weights [5]. However, ESN has a major problem, which is instability, and suffers from the ill-posed problem in the learning process, when the number of observations is less than the number of output neurons weights [5]. To tackle this problem, several extensions of ESN have been proposed [36,37,11,5,38–40].

In 2018 Chen and Lin [41] proposed the broad learning system (BLS), which is a randomized neural network with a flat structure [20,3]. The BLS has received considerable attention from the researchers due to its outstanding performance in different machine learning problems, especially time series prediction [20]. Xu et al. [3] developed a novel recurrent BLS (RBLS). To make the network capable of learning the dynamic characteristics of the time series, they add feedback loops in the enhancement nodes and used the conjugate gradient method to update the parameters of RBLS.

FNNs are a combination of ANNs and fuzzy logic systems (FLSs), which benefit from both the learning capability of ANNs and the semantic transparency and interpretability of FLSs [42,30,43]. Different types of FNNs have been proposed in the literature [44–48,26,27,30]. Angelov and Filev [44] developed eTS, which is an evolving Takagi–Sugeno (TS) model that employs recursive clustering with subtraction to update the network structure. Rong et al. [45] proposed the sequential adaptive fuzzy inference system (SAFIS), which implements a zero-order TS model and uses a rule influence metric to add or remove fuzzy rules. SAFIS updates its rules using an extended Kalman filter [30,45]. Subramanian and Suresh [46] introduced a meta-cognitive neuro-fuzzy inference system (McFIS) in 2012. They proposed a meta-cognitive sequential learning method for McFIS, which chooses the best training strategy for each sample based on its instantaneous error and spherical potential of the rule antecedents. PANFIS [47] is proposed by Pratama et al. in 2013. It starts the learning process with an empty fuzzy rule base and grows it by statistical contributions of the fuzzy rules. PANFIS also employs rule blending for pruning redundant rules [26,47,30]. Pratama et al. [48] extended PANFIS idea and proposed GENEFIS. In GENEFIS each feature's contribution is measured both in the antecedent and in the consequent of the rules. GENEFIS has the capability of online feature selection [48]. Ebadzadeh and Salimi-Badr [49] proposed CFNN, an FNN with correlated fuzzy rules, which uses the Levenberg–Marquardt (LM) method to learn fuzzy rules parameters. Their proposed method can approximate nonlinear functions with highly correlated input variables with fewer fuzzy rules. ICFNN [42], an FNN with interpretable correlated-contours fuzzy rules, was proposed in another paper by Ebadzadeh and Salimi-Badr. They introduced a novel shapeable membership function (MF) with an adjustable shape to form contours with different shapes. They also used the LM method to fine-tune fuzzy rules. However, since these models do not have recurrent connections, they could not learn temporal dependencies and memorize past information. As a result, the RFNNs were proposed.

Juang et al. [24] proposed RSEFNN-LF, a recurrent fuzzy neural network with local feedback. The RSEFNN-LF obtains its recurrent structure by locally feeding a fuzzy rule's firing strength back to itself. They used the Kalman filter and gradient descent algorithm for parameter learning. MRIT2NFS [25], a mutually recurrent interval type-2 neuro-fuzzy system, was proposed by Lin et al. MRIT2NFS uses interval type-2 fuzzy sets for the antecedent part of fuzzy rules. Its recurrent structure comes from a local internal feedback, which is established by feeding the firing strength of each rule to all rules, including itself. This RFNN employs type-2 fuzzy clustering for structure learning and the rule-ordered Kalman filter algorithm for parameter learning. Samanta et al. [26] developed a novel spatio-temporal fuzzy inference system (SPATFIS), which uses memory type neurons to incorporate spatial and temporal information of the time series. SPATFIS has a dual recurrent structure (input and defuzzification layers) and employs a novel learning method to add and remove fuzzy rules. Its stability is proved in [26]. However, a major drawback of SPATFIS is that sometimes its memory neurons cannot track rapidly changing system dynamics [30]. Samanta et al. [27] proposed NFIS-DN in 2019. NFIS-DN uses dynamic neurons, which consider only the impact of finite past observations, allowing the system to have finite memory. An evolving RFNN (i.e., eRIT2IFNN) was introduced by Luo et al. [28]. The eRIT2IFNN utilizes interval type-2 fuzzy sets to improve uncertainty modeling and employs Takagi–Sugeno-Kang (TSK) fuzzy rules for inference. Moreover, eRTI2IFNN uses a density-based clustering method for structure learning. Its recurrent structure comes from a local internal feedback loop, which is created by feeding the firing strength of each rule to itself. Ding et al. [29] developed SORFNN-MTSA, a self-organizing RFNN. SORFNN-MTSA employs a self-organization mechanism to optimize its structure. It utilizes a recurrent mechanism, based on wavelet transform and fuzzy Markov chain to increase the convergence speed. In [30], a Bayesian neuro-fuzzy inference system (BaNFIS) is proposed, which estimates temporal dependencies on past observations using an online Bayesian probabilistic method. BaNFIS only keeps past information as long as it is required and uses them globally or locally. Based on how the model uses past information, the authors in [30] proposed two models: Global BaNFIS and Local BaNFIS. For more details on these models, please refer to [30]. Table 1 summarizes the advantages and disadvantages of different time series prediction methods. As mentioned, when dealing with strong nonlinear problems, RFNN should be capable of learning multiple functions simultaneously, but most RFNNs learn a single function, so they can not generate different outputs for a specific input. This paper attempts to fill this gap by proposing a novel multi-functional recurrent fuzzy neural network, which can determine the system's state and learn multiple functions simultaneously by employing it.

3. Background

To clarify the proposed network and make the paper more compact, this section presents a brief introduction of particle swarm optimization (PSO). The PSO was proposed by Kennedy and Eberhart [50] in 1995, is a population-based stochastic optimization method inspired by the social behaviors observed in flocking birds. In PSO, a candidate solution of the optimization problem is referred to as a particle, and a group of particles makes a swarm. Each particle has its position and velocity. The ith particle's velocity and position at kth iteration are updated according to Eqs. (1) and (2).

$$\mathbf{v}_i^{k+1} = w\mathbf{v}_i^k + c_1\mathbf{r}_1\left(\mathbf{p}_i^k - \mathbf{x}_i^k\right) + c_2\mathbf{r}_2\left(\mathbf{p}_{gbest}^k - \mathbf{x}_i^k\right) \tag{1}$$

$$\mathbf{x}_i^{k+1} = \mathbf{x}_i^k + \mathbf{v}_i^{k+1} \tag{2}$$

where $\mathbf{x}_i^k$ and $\mathbf{v}_i^k$ denote the position and velocity of the ith particle at the kth iteration, respectively. $\mathbf{p}_i^k$ and $\mathbf{p}_{gbest}^k$ are the personal best position of the ith particle and the global best position of the swarm, respectively. w is the inertia weight. c_1 and c_2 are cognitive and social acceleration coefficients determining the relative impor-

Table 1
Summary of the advantages and disadvantages of different time series prediction methods.

Authors	Year	Model	Type	Advantages	Disadvantages
Li et al. [32]	2021	ECA-Adam-RBFNN	RBFNN	-Ability to learn the nonlinear relationship between input and output -Low training time -Having a low standard deviation due to its simplicity -Performs more robustly than multilayer perceptron	-Local minima problem -Determining the optimal structure and hyperparameters is difficult -Unable to model the temporal dependency of the time series data -Low accuracy in long-term prediction
Zhu and Meng [33]	2021	RBFNN-GA			
Han et al. [34]	2022	ASOL-SORBFNN			
Han et al. [37]	2014	L$_1$ESN	ESN		-Stability problems -Vanishing and exploding gradient problems -Collinearity problems when using high-dimensional reservoirs
Scardapane et al. [36]	2016	L$_1$ESN			
Xu et al. [11]	2016	AEESN			
Xu et al. [5]	2019	HESN			
Angelov and Filev [44]	2004	eTS	FNN	-Interpretability and semantic transparency -Ability to handle nonstochastic uncertainties -Having a good local representation power -Human-like reasoning capability	-Unable to model the temporal dependency of the time series data -Low accuracy in long-term prediction
Rong et al. [45]	2006	SAFIS			
Subramanian and Suresh [46]	2012	McFIS			
Pratama et al. [47]	2013	PANFIS			
Pratama et al. [48]	2013	GENEFIS			
Ebadzadeh and Salimi-Badr [49]	2015	CFNN			
Ebadzadeh and Salimi-Badr [42]	2017	ICFNN			
Juang et al. [24]	2010	RSEFNN-LF	RFNN	-Ability to learn temporal dependencies within time series -Interpretability and semantic transparency -Ability to handle nonstochastic uncertainties -Having a good local representation power	-Stability problems -Learning a single function for time series prediction task
Lin et al. [25]	2013	MRIT2NFS			
Samanta et al. [26]	2019	SPATFIS			
Samanta et al. [27]	2019	NFIS-DN			
Luo et al. [28]	2019	eRIT2IFNN			
Ding et al. [29]	2021	SORFNN-MTSA			
Subhrajit et al. [30]	2021	BaNFIS			

tance of $\mathbf{p}_i^k$ and $\mathbf{p}_{gbest}^k$. $\mathbf{r}_1$ and $\mathbf{r}_2$ are uniformly distributed random vectors within the interval $[0,1]$ [51].

4. Proposed method

In this section, the proposed multi-functional recurrent fuzzy neural network is presented. MFRFNN consists of two FNNs with TSK fuzzy rules. One produces the system's output (called output network), and the other determines the state of the system (called state network). These two networks connect with a feedback loop, which helps MFRFNN in memorizing historical information of past observations. Furthermore, the state network allows it to learn multiple functions simultaneously that result in capturing the dynamic characteristics of chaotic time series. An overview of the proposed method is shown in Fig. 2.

Let N denote the number of the states, K_1 and K_2 denote the number of fuzzy rules of the output network and state network, respectively. Then, the output network performs N function approximations, each with K_1 fuzzy rules, i.e., it learns a function for each state. The system's output consists of N segments. In each segment, a state approximates a function, and the final output is the sum of these functions. The state network also performs N function approximations, each with K_2 fuzzy rules, to determine the next state of the system. Fig. 3 demonstrates the structure of MFRFNN. As shown in Fig. 3, both networks consist of five layers. The operation function of the neurons in each layer is described as follows:

1. Input layer: This layer accepts input variables, and its neurons correspond to the membership functions (MFs). It is used to compute the membership values of input variables. Let $\mathbf{x} = [x_1, x_2, \cdots, x_d]^T$ denote the input and $\hat{y}$ denote the predicted output. $A_{i,j}$ and $B_{i,j}$ are the MFs for x_j in the ith rule of output and state network, respectively, and the membership value of jth input variable x_j on $A_{i,j}$ is denoted by $\mu_{A_{i,j}}(x_j)$, so the output of each neuron $A_{i,j}$ in this layer is the membership value of x_j on $A_{i,j}$, i.e., $\mu_{A_{i,j}}(x_j)$. Obvi-

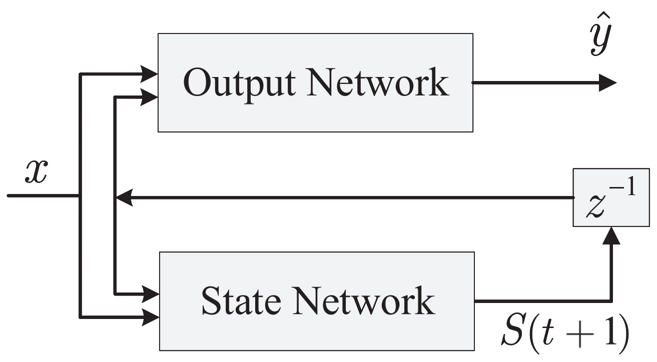


Fig. 2. Overview of MFRFNN.

ously, there are $K_1 \times d$ and $K_2 \times d$ neurons for the output and state network in this layer, respectively.

2. Fuzzy rules layer: Neurons at this layer represent the fuzzy rules, and their output represents the firing strength of a rule. Let r_i and q_i denote the output of the ith neuron of this layer for the output and state network, respectively. They can be computed by applying the T-norm operator on the previous layer's outputs. Using the algebraic product as the T-norm operator, the firing strength of each rule can be computed by (3) and (4).

$$r_i(\mathbf{x}) = \prod_{j=1}^{d} \mu_{A_{i,j}}(x_j) \tag{3}$$

$$q_i(\mathbf{x}) = \prod_{j=1}^{d} \mu_{B_{i,j}}(x_j) \tag{4}$$

3. Normalized fuzzy rules layer: Output of the neurons in this layer represents the normalized firing strength of each rule ($\bar{r}_i$ and $\bar{q}_i$ for the output and state network, respectively) and can be computed by (5) and (6).

$$\bar{r}_i(\mathbf{x}) = \frac{r_i(\mathbf{x})}{\sum_{j=1}^{K_1} r_j(\mathbf{x})} \tag{5}$$

$$\bar{q}_i(\mathbf{x}) = \frac{q_i(\mathbf{x})}{\sum_{j=1}^{K_2} q_j(\mathbf{x})} \tag{6}$$

4. Extended fuzzy rules layer: For each network, N linear combinations of normalized firing strength of rules are computed in this layer, and the output of N separate functions is determined. There are N neurons in this layer for each network. The output of the neurons in this layer represents the output of approximated functions (F_i and G_i for the output and state network, respectively) that can be computed as follows:

$$F_j = \sum_{i=1}^{K_1} \bar{r}_i w_{ij} \tag{7}$$

$$G_j = \sum_{i=1}^{K_2} \bar{q}_i v_{ij} \tag{8}$$

where w_{ij} and v_{ij} represent the link weight corresponding to the ith rule of the output and state network in the jth state of the system, respectively, the link weight matrix of the output network ($\mathbf{W}$) can be expressed by (9).

$$\mathbf{W} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ w_{K_1 1} & w_{K_1 2} & \cdots & w_{K_1 N} \end{bmatrix} \tag{9}$$

5. Output layer: Output layer of the output network computes the system's output ($\hat{y}$). In this layer, the output of functions from the previous layer is multiplied by state signals (from state net-

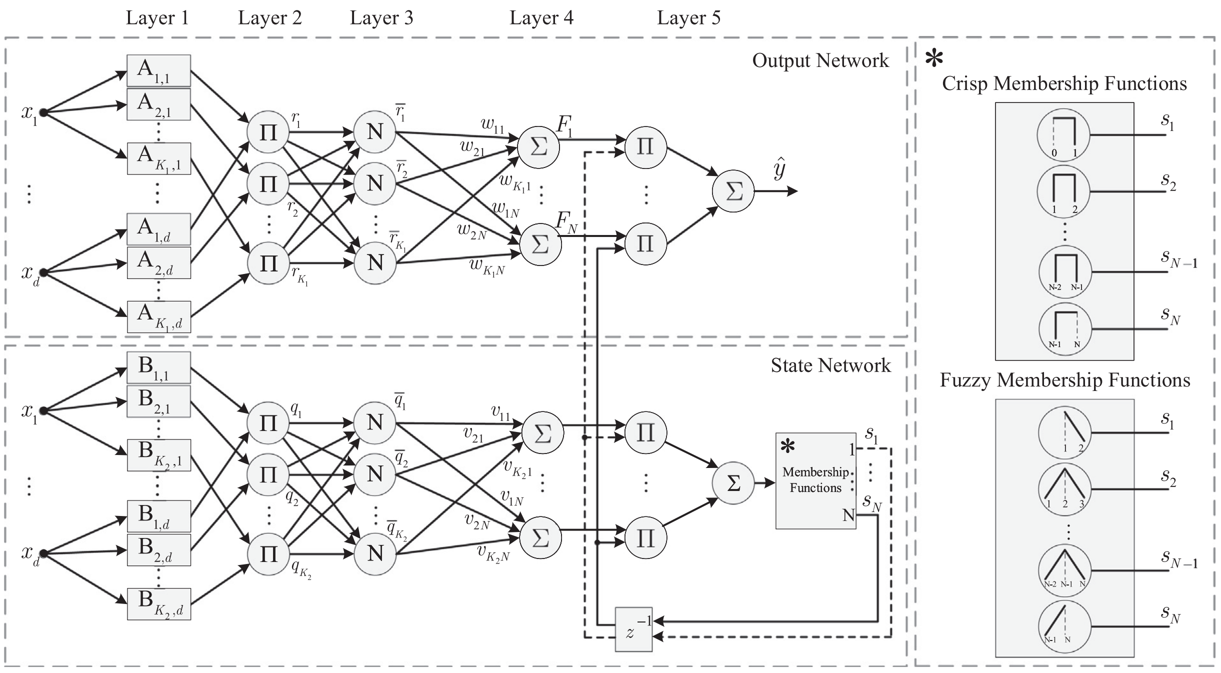


Fig. 3. MFRFNN architecture. The output network produces the system's output, and the state network produces state signals. Membership functions (MFs) in the output layer can be crisp or fuzzy MFs. If crisp MFs are used, the network's states become discrete, and if fuzzy MFs are used, its states become continuous.

work); as a result, the function corresponding to the current state is activated, and the other functions are deactivated (multiplied by zero) and have no effect on the output. The final output of the system is the sum of these functions. Let $\mathbf{F}(t)$ and $\mathbf{S}(t)$ denote vectors whose entries are the output of approximated functions and state signals of the system at the time step t, respectively, as presented in (10).

$$\begin{aligned}\mathbf{F}(t) &= [F_1, F_2, \cdots, F_N]^T \\ \mathbf{S}(t) &= [s_1, s_2, \cdots, s_N]^T\end{aligned} \tag{10}$$

The output of the system at the time step t, $(\hat{y})$ is computed as follows:

$$\hat{y}(t) = \mathbf{F}(t)^T \mathbf{S}(t) \tag{11}$$

Output layer of the state network produces state signals as output. In this layer, first, the output of neurons from layer 4 is multiplied by state signals. Then, the results are summed up and given as input to the membership functions. Then, the output of membership functions determines the next state of the system and considered as state signals. Finally, these signals go to a delay unit, and the output of the delay unit goes to layer 5 of both networks as a feedback loop. The intermediate output of this layer at the time step t is computed as follows:

$$o(t) = \mathbf{G}(t)^T \mathbf{S}(t) \tag{12}$$

where $\mathbf{G}(t)$ denotes a vector whose entries are the output of the previous layer's neurons, as presented in (13) and can be computed by (14).

$$\mathbf{G}(t) = [G_1, G_2, \cdots, G_N]^T \tag{13}$$

$$\mathbf{G}(t) = \mathbf{V}^T \mathbf{Q}(t) \tag{14}$$

where $\mathbf{V}$ and $\mathbf{Q}(t)$ denote the link weight matrix of the state network and vector of normalized firing strength of the fuzzy rules at the time step t, respectively, expressed in (15).

$$\mathbf{Q}(t) = \begin{bmatrix} \bar{q}_1 \\ \bar{q}_2 \\ \vdots \\ \bar{q}_{K_2} \end{bmatrix}, \mathbf{V} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1N} \\ v_{21} & v_{22} & \cdots & v_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ v_{K_2 1} & v_{K_2 2} & \cdots & v_{K_2 N} \end{bmatrix} \tag{15}$$

Assuming that all entries of $\mathbf{V}$ and $\mathbf{W}$ are in the range $[0, 1]$, then $o(t)$ is normalized in the range $[1, N]$ and gives as input to the MFs. The input to the MFs at the time step $t, \bar{o}(t)$ is computed as follows:

$$\bar{o}(t) = [o(t) \times (N-1)] + 1 \tag{16}$$

Let E_i denote the ith MF at the output layer, the system's state at the time step $t+1, \mathbf{S}(t+1)$ is determined by (17).

$$\mathbf{S}(t+1) = \begin{bmatrix} \mu_{E_1}(\bar{o}(t)) \\ \mu_{E_2}(\bar{o}(t)) \\ \vdots \\ \mu_{E_N}(\bar{o}(t)) \end{bmatrix} \tag{17}$$

where $\mu_{E_i}(\bar{o}(t))$ is the membership value of $\bar{o}(t)$ on E_i. As shown in Fig. 3, MFs can be crisp or fuzzy MFs. If crisp MFs are used, the network's states become discrete states, and if fuzzy MFs are used, its states become continuous states. In the case of fuzzy MFs (i.e., continuous states), the final output is a weighted sum of N approximated functions.

As above description, the total number of trainable parameters for MFRFNN is $(K_1 + K_2) \times N$. The link weight matrix of the output network ($\mathbf{W}$), and state network ($\mathbf{V}$) have $K_1 \times N$ and $K_2 \times N$ trainable parameters, respectively. The training procedure of the output network weight matrix is described as follows:

Given a training dataset $D = \{\mathbf{x}^{[t]}, y^{[t]}\}_{t=1}^{p}$, where p denotes the number of training samples. In addition to (11), the output of the system can also be computed by (18):

$$\hat{y} = \mathrm{tr}\left(\left(\mathbf{R}(t)\mathbf{S}^T(t)\right)^T \mathbf{W}\right) \tag{18}$$

where $\mathrm{tr}(\cdot)$ denotes the trace of a matrix and $\mathbf{R}(t)$ is a column vector whose entries are the normalized firing strength of fuzzy rules as presented in (19):

$$\mathbf{R}(t) = \left[\bar{r}_1, \bar{r}_2, \cdots, \bar{r}_{K_1}\right]^T \tag{19}$$

By plugging p training data into (18), the matrix Eq. (20) is obtained.

$$\mathbf{A}\boldsymbol{\theta} = \mathbf{y} \tag{20}$$

$$\begin{aligned}\mathbf{A} &= \begin{bmatrix} \bar{r}_1^{[1]} s_1 & \bar{r}_1^{[1]} s_2 & \cdots & \bar{r}_2^{[1]} s_1 & \cdots & \bar{r}_{K_1}^{[1]} s_N \\ \bar{r}_1^{[2]} s_1 & \bar{r}_1^{[2]} s_2 & \cdots & \bar{r}_2^{[2]} s_1 & \cdots & \bar{r}_{K_1}^{[2]} s_N \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \bar{r}_1^{[p]} s_1 & \bar{r}_1^{[p]} s_2 & \cdots & \bar{r}_2^{[p]} s_1 & \cdots & \bar{r}_{K_1}^{[p]} s_N \end{bmatrix} \\ \boldsymbol{\theta} &= [w_{11} \quad w_{12} \quad \cdots \quad w_{21} \quad \cdots \quad w_{K_1 N}]^T \\ \mathbf{y} &= \left[y^{[1]} \quad y^{[2]} \quad \cdots \quad y^{[p]}\right]^T\end{aligned} \tag{21}$$

where $\bar{r}_i^{[j]}$ denotes the normalized firing strength of ith rule for the jth training sample and $y^{[i]}$ denotes the actual output of the ith training sample. A closed-form solution for (20), which minimizes $\|\mathbf{A}\boldsymbol{\theta} - \mathbf{y}\|^2$ can be derived by the Moore–Penrose pseudoinverse:

$$\boldsymbol{\theta}^* = \left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{y} \tag{22}$$

Obviously, $\boldsymbol{\theta}^*$ is a column vector of length $(K_1 \times N)$, whose entries are the weight matrix ($\mathbf{W}$) entries.

Since the relationship between the link weight matrix of the state network ($\mathbf{V}$) and the output is not linear, the trainable parameters of $\mathbf{V}$ cannot be obtained by the linear least-squares method. Therefore, PSO is used for learning these parameters. In the proposed training algorithm, in addition to position ($\mathbf{x}$) and velocity ($\mathbf{v}$), each particle has its own output network's weight matrix ($\mathbf{w}$). Obviously, in this optimization problem, $\mathbf{x}$ represents the weight matrix of the state network that has to be optimized. The training algorithm of the output network weight matrix is summarized in Algorithm 1. The cost value calculation algorithm in the training phase is also implemented in this algorithm. MFRFNN training algorithm details are shown in Algorithm 2. Finally, the prediction algorithm in the test phase is summarized in Algorithm 3.

Algorithm 1: Training of output network weight matrix

```
Input: $K_1, K_2, N, \mathbf{V}$, Training dataset $D = \{\mathbf{x}^{[t]}, y^{[t]}\}_{t=1}^{p}$
Output: $\theta^*$, Cost Value $C$

$\mathbf{S} \leftarrow \mathbf{0}$
$\mathbf{S}_1 \leftarrow 1$ // Starting from state 1
$\mathbf{A} \leftarrow [\ ]$ // An empty matrix
for $t \leftarrow 1$ to $p$ do
    for $i \leftarrow 1$ to $K_1$ do
        Compute $r_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(3)
    end
    for $i \leftarrow 1$ to $K_1$ do
        Compute $\bar{r}_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(5)
    end
    Add new row $\left[\bar{r}_1^{[t]}s_1 \quad \bar{r}_1^{[t]}s_2 \cdots \bar{r}_1^{[t]}s_N \quad \bar{r}_2^{[t]}s_1 \cdots \bar{r}_{K_1}^{[t]}s_N\right]$ to
     matrix $\mathbf{A}$ in Eq. (21)
    for $i \leftarrow 1$ to $K_2$ do
        Compute $q_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(4)
    end
    for $i \leftarrow 1$ to $K_2$ do
        Compute $\bar{q}_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(6)
    end
    Compute $\mathbf{G}(t)$ using Eq. (14)
    Compute $o(t)$ using Eq. (12)
    Compute $\bar{o}(t)$ using Eq. (16)
    Compute $\mathbf{S}$ using Eq. (17) // Determine the next state
end
Compute $\theta^*$ using Eq. (22)
$\hat{\mathbf{y}} \leftarrow \mathbf{A}\theta^*$
$C \leftarrow \sqrt{\frac{1}{p}\sum_{i=1}^{p}\left(y^{[i]} - \hat{\mathbf{y}}_i\right)^2}$
```

Algorithm 3: Predicting the output in the test phase

```
Input: $K_1, K_2, N, \mathbf{V}, \theta^*$ Test dataset $D = \{\mathbf{x}^{[t]}, y^{[t]}\}_{t=1}^{p}$
Output: Predicted output ($\hat{\mathbf{y}}$)

$\mathbf{S} \leftarrow \mathbf{0}$
$\mathbf{S}_1 \leftarrow 1$ // Starting from state 1
$\mathbf{A} \leftarrow [\ ]$ // An empty matrix
for $t \leftarrow 1$ to $p$ do
    for $i \leftarrow 1$ to $K_1$ do
        Compute $r_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(3)
    end
    for $i \leftarrow 1$ to $K_1$ do
        Compute $\bar{r}_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(5)
    end
    Add new row $\left[\bar{r}_1^{[t]}s_1 \quad \bar{r}_1^{[t]}s_2 \cdots \bar{r}_1^{[t]}s_N \quad \bar{r}_2^{[t]}s_1 \cdots \bar{r}_{K_1}^{[t]}s_N\right]$ to
     matrix $\mathbf{A}$ in Eq. (21)
    for $i \leftarrow 1$ to $K_2$ do
        Compute $q_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(4)
    end
    for $i \leftarrow 1$ to $K_2$ do
        Compute $\bar{q}_i\left(\mathbf{x}^{[t]}\right)$ using Eq.(6)
    end
    Compute $\mathbf{G}(t)$ using Eq. (14)
    Compute $o(t)$ using Eq. (12)
    Compute $\bar{o}(t)$ using Eq. (16)
    Compute $\mathbf{S}$ using Eq. (17) // Determine the next state
end
$\hat{\mathbf{y}} \leftarrow \mathbf{A}\theta^*$
```

Algorithm 2: Training of MFRFNN

```
Input: $K_1, K_2, N$, Training dataset $D = \{\mathbf{x}^{[t]}, y^{[t]}\}_{t=1}^{p}$
Output: The global best particle ($gbest$)

for each particle $i$ do
    Initialize $\mathbf{x}_i \in \mathbb{R}^{(K_2 \times N) \times 1}$ and $\mathbf{v}_i \in \mathbb{R}^{(K_2 \times N) \times 1}$ to random vectors
    Initialize $\mathbf{w}_i \in \mathbb{R}^{(K_1 \times N) \times 1}$ to a random vector
    $\mathbf{p}_i \leftarrow \mathbf{x}_i$
    Update $\mathbf{w}_i$ and calculate the cost value $f(\mathbf{p}_i)$ using Algorithm 1
end
$gbest \leftarrow \arg\min_i f(\mathbf{p}_i)$
while stopping criterion is not met do
    for each particle $i$ do
        Update $\mathbf{v}_i$ using Eq. (1)
        Update $\mathbf{x}_i$ using Eq. (2)
        Update $\mathbf{w}_i$ and calculate $f(\mathbf{x}_i)$ using Algorithm 1 ($\mathbf{w}_i \leftarrow \theta^*$)
        if $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ then
            $\mathbf{p}_i \leftarrow \mathbf{x}_i$
            if $f(\mathbf{p}_i) < f\left(\mathbf{p}_{gbest}\right)$ then
                $gbest \leftarrow i$
            end
        end
    end
end
```

5. Experimental results

This section evaluates the performance of MFRFNN on two chaotic systems (i.e., Lorenz and Rossler) and four real-world datasets, including Box–Jenkins Gas Furnace, Wind Speed Prediction, Google Stock Price Prediction, and Air Quality Index Prediction. These benchmarks are standard benchmarks and have been widely used in the time series prediction community. Since the mentioned datasets did not contain any outliers, Min–max normalization was used to scale all data into the range $[0, 1]$ in all experiments. To handle outliers, Z-score normalization (Standardization) can be used. The experiments were run on an Intel Core i5-8250U, 1.60 GHz CPU with 8 GB RAM, running Windows 10 operating system. To assess the performance of MFRFNN and compare its performance with other methods, five evaluation metrics were used: Mean Square Error (MSE), Root Mean Square Error (RMSE), Normalized Root Mean Square Error (NRMSE), Mean Absolute Error (MAE), and Symmetric Mean Absolute Percentage Error (SMAPE).

The parameters of MFRFNN, including the number of fuzzy rules for the output network and state network, number of states of the state network, and maximum number of fitness evaluations for the PSO algorithm, were chosen by trial and error using the validation set in each benchmark. For the number of fuzzy rules of the output and state network, different values from 2 to 5 were tested. For the number of states, different values from 2 to 15 were evaluated, and the maximum number of fitness evaluations for the PSO algorithm was chosen from the set $\{250, 500, 1000, 2000, 4000\}$. Moreover, the validation set was used to avoid overfitting and improve the

Table 2
The main parameters of MFRFNN in each benchmark.

Benchmark	K_1	K_2	N	Maximum Number of FES (PSO Algorithm)	Number of input steps
Lorenz System	27	27	3	500	1
Rossler System	27	27	3	250	1
Box-Jenking Gas Furnace	9	4	2	4000	1
Wind Speed Prediction	4	4	2	4000	1
Stock Price Prediction	3	3	2	4000	1
Air Quality Index Prediction	16	16	2	250	4

model's generalization. In each experiment, first, we chose the parameters' values based on the validation error and then set these parameters and computed the test error. Table 2 summarizes the selected parameters for each benchmark. To ensure a fair comparison with other methods, we set the number of input steps equal to the number of input steps of the comparing method. The direct forecasting strategy was used for all methods in all experiments. Also, symmetrical and uniformly distributed triangular MFs were used for the input layer of MFRFNN's output network and state network. It is worth mentioning that we evaluated different types of MFs, including triangular, Gaussian, combination of two Gaussians, and Generalized bell-shaped, and the best results were obtained by triangular MFs. Moreover, it has been proved theoretically in [52] that why in practice, this type of MFs work so well. We considered two cases for the state network's output layer's MFs: crisp MFs, i.e., MFRFNN with discrete states, and fuzzy MFs, i.e., MFRFNN with continuous states.

5.1. Lorenz system

The Lorenz system is a non-linear, three-dimensional system that can be described as follows:

$$\begin{aligned} dx/dt &= \sigma(y-x) \\ dy/dt &= x(\rho - z) - y \\ dz/dt &= xy - \beta z \end{aligned} \tag{23}$$

when $\sigma = 10, \beta = 8/3$, and $\rho = 28$, the system has chaotic solutions. The experimental setup, same as [3], was used in this paper, and the fourth-order Runge–Kutta method was used to generate samples. Table 3 summarizes the details of the experimental setup.

To evaluate the performance of MFRFNN on chaotic systems, this experiment was conducted using two state-of-the-art methods, including deep autoencoder (DAE) [53] and RBLS [3], as well as three other machine learning methods: ELM [54], ESN [35], and ε-SVR [55]. For this experiment, the parameters settings of other methods and experimental setup were the same as [3]. Table 4 presents the one-step-ahead prediction results of the Lorenz time series, including twenty independent runs' averages and standard deviations. Furthermore, to evaluate whether the superiority of a method was statistically significant, a two-tailed Welch's t-test with a significance level $\alpha = 0.05$ was applied for RMSE between MFRFNN with continuous states and other methods. Welch's t-test is a nonparametric univariate statistical test, useful when the two samples have unequal variances [56]. The last column of Table 4 shows the p-value of the two-tailed Welch's t-test. In all comparisons except the comparison of MFRFNN with continuous states and RBLS on the $z(t)$ series, the null hypothesis is clearly rejected based on the tests with a 95% confidence level ($p-\text{value} < 0.05$), giving statistically significant results. Fig. 4 shows one-step-ahead prediction curves, error curves, and histograms of errors for the Lorenz series generated by MFRFNN with continuous states.

Table 3
Details of the experimental setup for the Lorenz system.

Parameter	Value
Number of samples	20000
Initial state	$[12, 2, 9]$
Step size	0.01
Number of training samples	11250
Number of validation samples	3750
Number of test samples	5000

5.2. Rossler system

The Rossler system is a classical system, consisted of three nonlinear ordinary differential equations and can be defined by:

$$\begin{aligned} dx/dt &= -y - z \\ dy/dt &= x + ay \\ dz/dt &= b + z(x - c) \end{aligned} \tag{24}$$

when $a = 0.15, b = 0.2$, and $c = 10$, the system shows chaotic behavior. To compare the performance of MFRFNN with other methods under the same condition, the experimental setup, same as [5], was used for the Rossler system. In this setup, the fourth-order Runge–Kutta method was employed for sample generation. Some of the samples were discarded to eliminate the transient influence of the initial condition. Table 5 presents the details of the experimental setup for the Rossler system.

To evaluate the performance of MFRFNN on long-term prediction task, we compared its performance with six extensions of ESN: ESN based on L_1-norm (L_1ESN) [37], ESN based on L_2-norm regularization (L_2ESN) [38], ESN based on elastic net regularization (EESN) [39], ESN based on $L_{1/2}$ regularization ($L_{1/2}$ESN) [40], adaptive elastic ESN (AEESN) [11], and hybrid regularized ESN (HESN) [5]. The parameters settings of the mentioned algorithms were the same as [5], and the results were averaged over twenty runs, again similar to [5]. In chaotic time series, the largest predictable horizon is relevant to the largest Lyapunov exponent [5]. The chaotic time series' predictable horizon was estimated using the inverse of the largest Lyapunov exponent as expressed by (25).

$$\eta_{\max} = \frac{1}{l_{\max}} \tag{25}$$

where $l_{\max}$ denotes the largest Lyapunov exponent and $\eta_{\max}$ denotes the predictable horizon [5]. Same as [5], we used the Wolf method to compute $l_{\max}$. For the Rossler system, $l_{\max}$ is 1.18, so the predictable horizon can be computed using (25): $\eta_{\max} = 1/1.18 \approx 0.847$. Since the step size for the Rossler system is set to 0.03, the predictable step is $0.847/0.03 \approx 28$. So, for Rossler system, the predictable horizons were considered from 1 to 28. Table 6 shows multi-step ahead prediction results of the Rossler time series.

Table 4
One-step-ahead prediction error comparison on the Lorenz System.

Series	Method		RMSE	SMAPE	NRMSE	*p*-value
Lorenz System $x(t)$	DAE	avg	8.05E−04	3.90E−05	1.02E−04	5.35E−31
		std	(2.91E−05)	(3.84E−06)	(3.68E−06)	
	ESN	avg	2.47E−04	1.23E−05	3.13E−05	7.94E−49
		std	(6.45E−06)	(3.92E−07)	(8.18E−07)	
	ELM	avg	7.21E−04	3.42E−05	9.14E−05	2.75E−41
		std	(8.94E−07)	(6.27E−08)	(1.13E−07)	
	ε-SVR	avg	3.58E−03	2.20E−04	4.54E−04	1.49E−53
		std	(0.00E+00)	(0.00E+00)	(0.00E+00)	
	RBLS	avg	2.05E−04	9.30E−06	2.60E−05	3.73E−32
		std	(1.83E−06)	(9.10E−08)	(2.32E−07)	
	MFRFNN	avg	6.52E−05	2.58E−06	8.42E−06	8.25E−11
	(Discrete States)	std	(1.59E−05)	(4.46E−07)	(2.06E−06)	
	MFRFNN	avg	**2.44E−05**	**7.27E−07**	**3.15E−06**	–
	(Continuous States)	std	(6.62E−06)	(2.00E−07)	(8.56E−07)	
Lorenz System $y(t)$	DAE	avg	2.42E−03	9.02E−05	2.69E−04	3.67E−52
		std	(4.87E−05)	(5.93E−06)	(5.43E−06)	
	ESN	avg	5.65E−04	2.61E−05	6.29E−05	3.98E−18
		std	(1.75E−05)	(9.85E−07)	(1.95E−06)	
	ELM	avg	2.26E−03	8.10E−05	2.51E−04	9.31E−34
		std	(1.77E−06)	(1.28E−07)	(1.97E−07)	
	ε-SVR	avg	5.86E−03	3.29E−04	6.52E−04	2.04E−42
		std	(0.00E+00)	(0.00E+00)	(0.00E+00)	
	RBLS	avg	4.15E−04	1.71E−05	4.61E−05	3.26E−06
		std	(6.16E−06)	(2.23E−07)	(6.86E−07)	
	MFRFNN	avg	1.53E−03	6.62E−05	1.74E−04	2.33E−16
	(Discrete States)	std	(2.15E−04)	(9.85E−06)	(2.44E−05)	
	MFRFNN	avg	**3.58E−04**	**1.40E−05**	**4.05E−05**	–
	(Continuous States)	std	(3.95E−05)	(2.68E−06)	(4.49E−06)	
Lorenz System $z(t)$	DAE	avg	1.98E−03	2.34E−05	2.29E−04	3.13E−42
		std	(2.79E−05)	(1.16E−06)	(3.24E−06)	
	ESN	avg	6.16E−04	8.56E−06	7.16E−05	4.43E−14
		std	(1.90E−05)	(2.75E−07)	(2.21E−06)	
	ELM	avg	1.78E−03	2.04E−05	2.07E−04	5.07E−29
		std	(7.39E−07)	(3.19E−08)	(8.58E−08)	
	ε-SVR	avg	4.79E−03	8.23E−05	5.57E−04	1.05E−38
		std	(0.00E+00)	(0.00E+00)	(0.00E+00)	
	RBLS	avg	4.52E−04	5.38E−06	5.25E−05	1.65E−01
		std	(7.69E−06)	(7.97E−08)	(8.93E−07)	
	MFRFNN	avg	1.29E−03	1.62E−05	1.51E−04	1.28E−17
	(Discrete States)	std	(1.53E−04)	(1.57E−06)	(1.80E−05)	
	MFRFNN	avg	**4.36E−04**	**4.79E−06**	**5.09E−05**	–
	(Continuous States)	std	(4.90E−05)	(5.65E−07)	(5.71E−06)	

Fig. 5 shows the relationship between prediction errors and prediction horizons for different methods.

5.3. Box–Jenkins gas furnace problem

Box–Jenkins gas furnace is a well-known time series forecasting problem in which the output CO_2 concentration is predicted using the input oxygen flow rate [30]. Same as [30], the forecasting problem can be represented by (26).

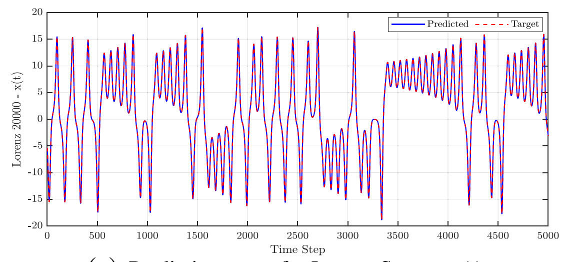
$$\hat{y}(t) = f(y(t-1), u(t)) \tag{26}$$

Where $u(t)$ and $y(t)$ denote the oxygen flow rate and CO_2 concentration rate, respectively. The dataset consists of 290 samples, 200 of which were used as the training set, and the remaining 90 samples were used as the test set. To assess the performance of MFRFNN on real-world time series, we compared its performance with three state-of-the-art RFNNs, including NFIS-DN [27], SPATFIS [26], and BaNFIS (global BaNFIS and local BaNFIS) [30]. Moreover, five other FNNs were used for comparison: eTS [44], SAFIS [45], McFIS [46], PANFIS [47], and GENEFIS [48]. Table 7 presents the one-step-ahead prediction results of the Box–Jenkins gas furnace problem.

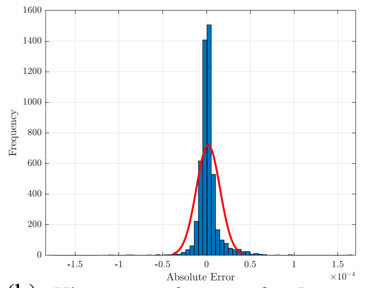
5.4. Wind speed prediction problem

The wind speed prediction problem is a non-linear, dynamic, and volatile problem in which the future value of wind speed is predicted using the current wind speed and wind direction. The dataset is obtained from the Iowa Department of Transport's website.[1] The data was collected from the Washington station during a one-month period (February 2011), sampled every ten minutes, and averaged hourly. There are 500 samples in the training set and 1000 samples in the test set [30]. This dataset is more challenging than the Box–Jenkins dataset due to the existence of noise. This experiment compared the proposed method's performance with the same algorithms we used in the Box–Jenkins dataset. The results are given in Table 8.

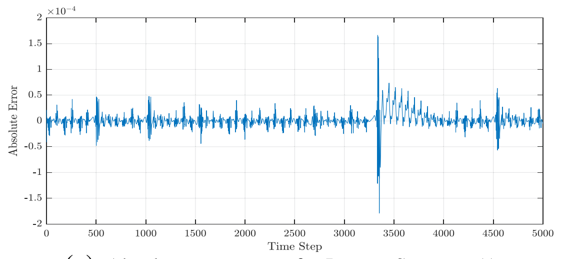
[1] http://mesonet.agron.iastate.edu/request/awos/1min.php



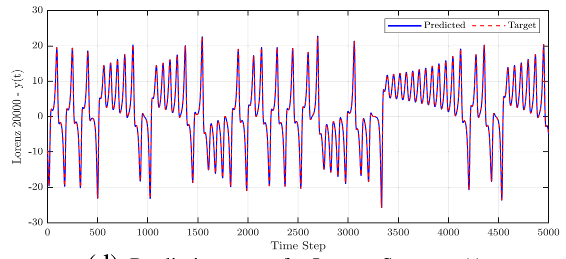
(a) Prediction curve for Lorenz System $x(t)$



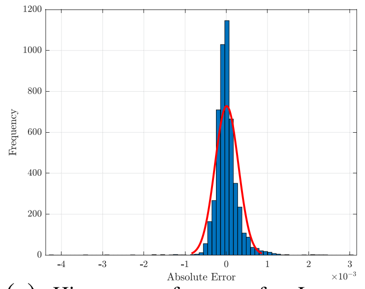
(b) Histogram of errors for Lorenz System $x(t)$



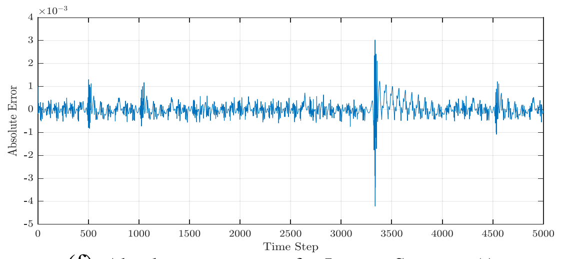
(c) Absolute error curve for Lorenz System $x(t)$



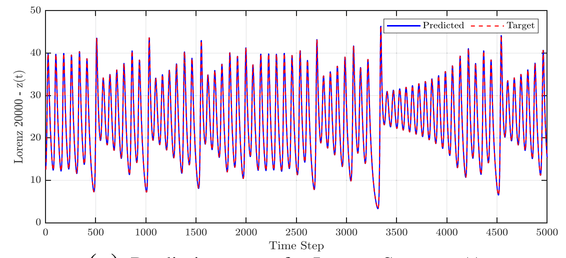
(d) Prediction curve for Lorenz System $y(t)$



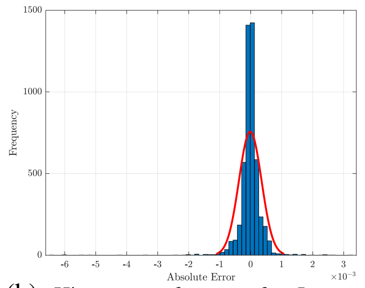
(e) Histogram of errors for Lorenz System $y(t)$



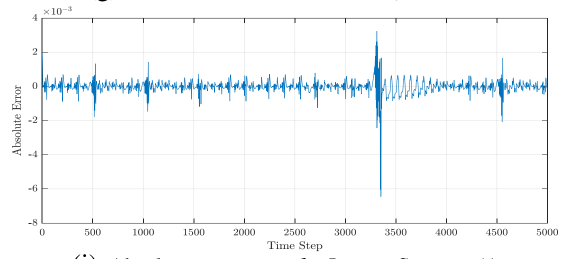
(f) Absolute error curve for Lorenz System $y(t)$



(g) Prediction curve for Lorenz System $z(t)$



(h) Histogram of errors for Lorenz System $z(t)$



(i) Absolute error curve for Lorenz System $z(t)$

Fig. 4. One-step-ahead prediction curve, error curve, and histogram of errors for the Lorenz time series generated by MFRFNN.

Table 5
Details of the experimental setup for the Rossler system.

Parameter	Value
Number of samples	12700
Initial state	$[1, 1, 1]$
Step size	0.03
Number of discarded samples	7700
Number of training samples	3000
Number of validation samples	1000
Number of test samples	1000

5.5. Stock price prediction problem

Stock price prediction is a non-linear and highly volatile problem. In this problem, the future value of Google stock price is predicted using the current price as defined by (27).

$$\hat{y}(t) = f(y(t-1)) \tag{27}$$

The dataset was obtained from Yahoo Finance[2] during a six-year period from 19-August-2004 to 21-September-2010 as in [30]. The training set consisted of 1529 samples, and the test set 900 samples. To evaluate the performance of MFRFNN on another real-world time series, we compared its performance with the same RFNNs and FNNs used in Box–Jenkins and wind speed prediction datasets. Table 9 shows the one-step-ahead prediction results of the Google stock price prediction problem.

5.6. Air quality index prediction problem

In this experiment, we employed the air quality index (AQI) dataset [57] to evaluate the performance of MFRFNN in a real-world multi-step ahead prediction task. The AQI dataset was obtained from 12 observing stations around Beijing from 2013 to 2017, containing extremely frequent and drastic fluctuations. The dataset consisted of 35,064 samples, which were collected hourly for 1461 days. Each sample comprised six major pollution components, including fine particulate matter ($PM_{2.5}$), respirable particulate matter (PM_{10}), sulfur dioxide (SO_2), nitrogen dioxide (NO_2), carbon monoxide (CO), and ozone (O_3) [58,57]. Same as [58], the first 22800 samples (950 days) were used as the training set, 1200 samples (50 days) as the validation set, and 1200 samples as the test set. Also, four input steps were used as the input sequence.

In order to sustain the effectiveness of the proposed method in time series prediction, we compared its performance with five traditional machine learning models, including decision tree (DT), random forest (RF), support vector regression (SVR), multilayer perceptron (MLP), and long short-term memory (LSTM). We also considered some extensions of LSTM, including the nested LSTM (NLSTM) [59] and the stacked LSTM (SLSTM) [60], for comparison. Moreover, MFRFNN was compared with hybrid models incorporating various LSTM extensions and pre-processing methods, such as empirical mode decomposition (EMD) [61], variational mode decomposition (VMD) [62], and wavelet transform (WT) [63]. Furthermore, MTMC-NLSTM [58] was used as a state-of-the-art model for comparison. The experimental setup and parameters used for the mentioned methods were the same as [58]. Twenty independent runs were performed for each method, and the averages and standard deviations were reported in Tables 10–13. Table 10 and 11 present the five-step-ahead prediction results of the AQI dataset. Ten-step-ahead prediction results of the AQI dataset were reported in Table 12 and Table 13. Moreover, to evaluate whether the performance of a method was statistically significant, a two-tailed Welch's t-test with a 0.05 significance level was applied. The Welch's t-test was applied for RMSE between MFRFNN and other methods, and the obtained p-value was reported. As can be seen, all the results were statistically significant. Table 14 compares the number of parameters and average training time of different methods.

5.7. Sensitivity analysis

We performed a sensitivity analysis in this section to further strengthen this study. The parameters used in sensitivity analysis are as follows: Number of states and number of MFs in each dimension. The number of MFs was considered equal for both networks (i.e., output network and state network), and from 2 to 5 in each dimension. The number of states ranged from 1 to 15, so 60 independent experiments were performed, and the NRMSE was computed for MFRFNN with continuous states. The Lorenz time series and Box–Jenkins gas furnace problem were used as the benchmark in the aforementioned experiments. Fig. 6 shows the relationship between NRMSE, the number of MFs, and the number of states for the one-step-ahead prediction of the Lorenz time series. As can be seen, when MFRFNN employed one state, the NRMSE was high, because in this case, MFRFNN had no feedback loop, so it can not memorize historical information of the time series. In Fig. 6, when the number of states increased from 2 to 9, MFRFNN did not have high sensitivity to parameters, but when the number of states exceeded 9, NRMSE tended to increase with the increase of states. Moreover, when the number of states was large (i.e., ten and more), the performance strongly depended on the number of MFs and the number of fuzzy rules. Table 15 presents the NRMSE of the various number of states and MFs for the one-step-ahead prediction of the Box–Jenkins gas furnace problem. As can be seen, when the number of states and MFs was low, MFRFNN did not have high sensitivity to parameters. As the number of states and MFs increased, the number of parameters of the model increased, and the model's generalization started to deteriorate. As a result, the NRMSE on the test set increased.

5.8. Ablation study

In this section, we present an ablation study to demonstrate the impact of feedback loop between two networks (i.e., output network and state network) in the structure of MFRFNN with continuous states. We repeated the experiments on Lorenz and Rossler by MFRFNN without a feedback loop to conduct this study. In other words, we used the proposed structure with the same number of fuzzy rules but with just one state. We also evaluated MFRFNN without a feedback loop on real-world datasets. For the AQI dataset, the models were evaluated by a ten-step-ahead prediction task. To assess whether the results were statistically significant, a two-tailed Welch's $t-$test with a 0.05 significance level was applied for RMSE, and the obtained p-value was reported. Table 16 compares MFRFNN's performance with and without a feedback loop on the Lorenz and Rossler time series. Note that in the experiments with a feedback loop, the experimental setups were the same as in Table 2. As can be seen in Table 16, MFRFNN with feedback loop obtained lower RMSE and SMAPE in all experiments. Also, based on the statistical tests, the p-value was lower than the specified significance level (i.e., 0.05), giving statistically significant results. Table 17 presents the comparison of MFRFNN's performance with and without a feedback loop on real-world datasets. As shown in Table 17, MFRFNN with a feedback loop obtained lower RMSE and MAE in all experiments except AQI-PM_{10} and AQI-SO_2. For AQI-PM_{10}, the result was not considered statistically significant based on the statistical test. For AQI-SO_2,

[2] http://finance.yahoo.com

Table 6
Multi-step-ahead prediction RMSE comparison on the Rossler System-$x(t)$ series.

horizon	L_1ESN	L_2ESN	EESN	$L_{1/2}$ESN	AEESN	HESN	MFRFNN (Discrete States)		MFRFNN (Continuous States)	
							avg	std	avg	std
1	7.20E−03	7.60E−03	6.60E−03	7.90E−03	6.90E−03	5.40E−03	3.98E−04	6.97E−04	**2.65E−04**	3.63E−04
2	1.34E−02	1.35E−02	1.44E−02	1.56E−02	1.28E−02	1.05E−02	**5.05E−04**	5.03E−04	5.77E−04	7.09E−04
3	1.95E−02	2.05E−02	2.02E−02	2.88E−02	2.01E−02	1.59E−02	1.43E−03	1.03E−03	**8.80E−04**	5.49E−04
4	2.69E−02	2.92E−02	2.63E−02	4.79E−02	2.89E−02	2.23E−02	3.14E−03	2.06E−03	**1.52E−03**	6.60E−04
5	3.66E−02	4.01E−02	3.54E−02	7.09E−02	4.03E−02	3.05E−02	3.36E−03	8.84E−04	**3.04E−03**	1.32E−03
6	5.03E−02	5.47E−02	4.71E−02	9.91E−02	5.47E−02	4.14E−02	6.48E−03	4.57E−03	**4.40E−03**	1.43E−03
7	6.78E−02	7.36E−02	6.25E−02	1.31E−01	7.29E−02	5.57E−02	7.24E−03	1.81E−03	**6.94E−03**	2.10E−03
8	9.06E−02	9.77E−02	8.19E−02	1.67E−01	9.49E−02	7.38E−02	1.02E−02	2.58E−03	**9.50E−03**	3.20E−03
9	1.17E−01	1.27E−01	1.06E−01	2.07E−01	1.21E−01	9.61E−02	**1.23E−02**	1.75E−03	1.25E−02	3.06E−03
10	1.50E−01	1.62E−01	1.34E−01	2.49E−01	1.52E−01	1.23E−01	2.25E−02	1.33E−02	**1.79E−02**	7.99E−03
11	1.85E−01	2.03E−01	1.66E−01	2.94E−01	1.86E−01	1.53E−01	2.62E−02	1.01E−02	**1.91E−02**	4.73E−03
12	2.26E−01	2.48E−01	2.02E−01	3.41E−01	2.24E−01	1.87E−01	3.24E−02	9.49E−03	**2.78E−02**	1.09E−02
13	2.70E−01	2.99E−01	2.42E−01	3.90E−01	2.66E−01	2.25E−01	4.71E−02	2.59E−02	**3.31E−02**	6.81E−03
14	3.19E−01	3.54E−01	2.85E−01	4.41E−01	3.11E−01	2.66E−01	5.09E−02	3.27E−02	**3.56E−02**	5.42E−03
15	3.69E−01	4.13E−01	3.31E−01	4.92E−01	3.59E−01	3.10E−01	7.78E−02	6.15E−02	**4.78E−02**	1.37E−02
16	4.24E−01	4.75E−01	3.80E−01	5.45E−01	4.08E−01	3.56E−01	8.19E−02	4.19E−02	**5.24E−02**	9.36E−03
17	4.81E−01	5.40E−01	4.31E−01	5.98E−01	4.60E−01	4.03E−01	9.17E−02	5.37E−02	**6.93E−02**	1.38E−02
18	5.40E−01	6.06E−01	4.84E−01	6.51E−01	5.12E−01	4.52E−01	1.15E−01	7.04E−02	**8.14E−02**	1.02E−02
19	6.01E−01	6.73E−01	5.38E−01	7.04E−01	5.65E−01	5.01E−01	1.54E−01	1.46E−01	**8.50E−02**	1.13E−02
20	6.64E−01	7.40E−01	5.92E−01	7.56E−01	6.17E−01	5.51E−01	1.74E−01	1.03E−01	**1.01E−01**	1.73E−02
21	7.27E−01	8.07E−01	6.47E−01	8.08E−01	6.69E−01	5.99E−01	1.90E−01	1.30E−01	**1.20E−01**	2.91E−02
22	7.93E−01	8.72E−01	7.02E−01	8.59E−01	7.19E−01	6.47E−01	2.05E−01	1.47E−01	**1.48E−01**	4.23E−02
23	8.58E−01	9.35E−01	7.56E−01	9.10E−01	7.68E−01	6.93E−01	2.35E−01	1.55E−01	**1.87E−01**	8.19E−02
24	9.25E−01	9.95E−01	8.10E−01	9.60E−01	8.14E−01	7.37E−01	3.64E−01	2.62E−01	**2.49E−01**	1.16E−01
25	9.91E−01	1.05E+00	8.62E−01	1.01E+00	8.58E−01	7.79E−01	3.82E−01	2.51E−01	**3.20E−01**	1.82E−01
26	1.06E+00	1.10E+00	9.12E−01	1.06E+00	8.99E−01	8.19E−01	5.17E−01	2.97E−01	**3.43E−01**	1.56E−01
27	1.12E+00	1.15E+00	9.61E−01	1.10E+00	9.37E−01	8.55E−01	5.32E−01	2.84E−01	**3.75E−01**	2.63E−01
28	1.19E+00	1.20E+00	1.01E+00	1.15E+00	9.72E−01	8.89E−01	5.59E−01	2.68E−01	**4.43E−01**	2.16E−01

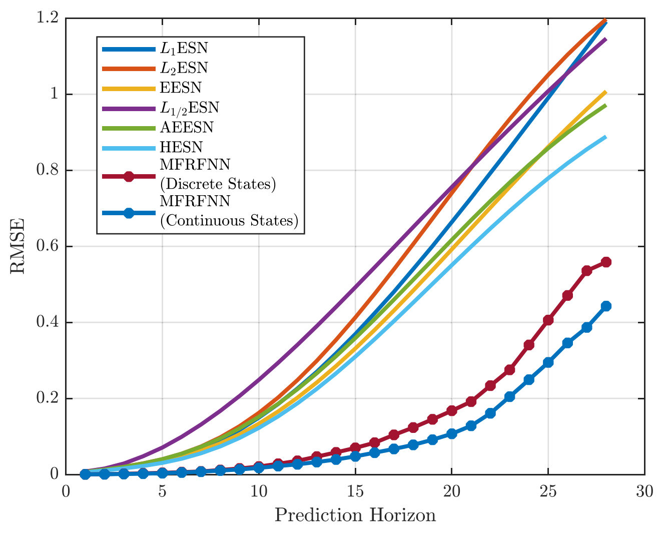


Fig. 5. Relationship between prediction errors and prediction horizons for different methods (Rossler System).

MFRFNN without a feedback loop obtained lower MAE. This may be because of the simplicity of the prediction task in this experiment. As a result, MFRFNN with one state (i.e., without a feedback loop) obtained better generalization and lower MAE. Note that MFRFNN with a feedback loop used only two states in this experiment. Based on the results, it is obvious that using the feedback loop and multiple states in the proposed method was substantially beneficial.

Table 7
One-step-ahead prediction error comparison on the Box–Jenkins gas furnace problem.

Method		RMSE	MSE	MAE
eTS		0.049	0.002	0.034
SAFIS		0.071	0.005	0.047
McFIS		0.045	0.002	0.028
PANFIS		0.070	0.005	0.048
GENEFIS		0.050	0.003	0.034
NFIS-DN		0.046	0.002	0.033
SPATFIS		0.050	0.003	0.036
Global BaNFIS		0.043	0.002	0.029
Local BaNFIS		0.063	0.004	0.049
MFRFNN	avg	0.039	0.002	0.028
(Discrete States)	std	(4.53E−04)	(3.52E−05)	(6.09E−04)
MFRFNN	avg	**0.037**	**0.001**	**0.026**
(Continuous States)	std	1.48E−03	(1.10E−04)	(1.04E−03)

Table 8
One-step-ahead prediction error comparison on the wind speed prediction problem.

Method		RMSE	MSE	MAE
eTS		0.380	0.144	0.262
SAFIS		0.376	0.141	0.257
McFIS		0.230	0.052	0.165
PANFIS		0.190	0.036	0.131
GENEFIS		0.153	0.023	0.105
NFIS-DN		0.150	0.022	0.106
SPATFIS		0.146	0.021	0.101
Global BaNFIS		0.136	0.018	0.099
Local BaNFIS		0.133	0.017	0.097
MFRFNN	avg	0.070	**0.005**	**0.048**
(Discrete States)	std	(1.57E−03)	(2.19E−04)	(1.03E−02)
MFRFNN	avg	**0.067**	**0.005**	**0.048**
(Continuous States)	std	(8.96E−04)	(1.21E−04)	(1.17E−03)

Table 9
One-step-ahead prediction error comparison on the Google stock price prediction problem.

Method		RMSE	MSE	MAE
eTS		0.070	0.005	0.047
SAFIS		0.071	0.005	0.051
McFIS		0.036	0.001	0.026
PANFIS		0.049	0.002	0.034
GENEFIS		0.036	0.001	0.025
NFIS-DN		0.030	0.001	0.019
SPATFIS		0.020	0.0004	0.015
Global BaNFIS		**0.016**	**0.0003**	**0.011**
Local BaNFIS		0.044	0.002	0.037
MFRFNN	avg	0.017	**0.0003**	0.012
(Discrete States)	std	(4.03E−04)	(1.39E−05)	(3.32E−04)
MFRFNN	avg	0.017	**0.0003**	0.012
(Continuous States)	std	(4.02E−04)	(1.39E−05)	(2.11E−04)

5.9. Comparison of different metaheuristic algorithms

As mentioned, the link weight matrix of the state network can not be obtained by the linear least-squares method. So, MFRFNN uses PSO to learn parameters of **V**. However, any other metaheuristic algorithm can be used instead of PSO. In this section, we employed different metaheuristic algorithms to train MFRFNN. These algorithms include Artificial Bee Colony (ABC) [64], Ant Colony Optimization for Continuous Domains (ACOR) [65], Bees Algorithm (BA) [66], Biogeography-based Optimization (BBO) [67], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [68], Differential Evolution (DE) [69], Firefly Algorithm (FA) [70], Genetic Algorithm (GA) [71], Harmony Search (HA) [72], Imperialist Competitive Algorithm (ICA) [73], Invasive Weed Optimization (IWO) [74], and Teaching–Learning-based Optimization (TLBO) [75]. As mentioned, all entries of **V** are in the range $[0, 1]$, so for all decision variables optimized by metaheuristic algorithms, the lower bound is 0, and the upper bound is 1. Table 18 shows the ten-step-ahead prediction results of AQI-SO_2 and the training time of each method.

5.10. State functions

In this section, the functions learned by each state are illustrated to analyze the role of states in MFRFNN. The experiment was conducted on the wind speed prediction dataset using the parameters of Table 2. MFRFNN with continuous states was used for this experiment. Fig. 7 shows the functions learned by each state, the one-step-ahead prediction curve, and the target curve. As can be seen, each state learned a different function, and the target output is a weighted sum of these functions.

6. Discussion

In this paper, we evaluated the performance of MFRFNN on six different benchmarks. The results of Lorenz time series experiments indicated that MFRFNN with continuous states outperformed other methods in all three series. After this method, MFRFNN with discrete states outperformed other methods in the $x(t)$ series. The prediction accuracy of RBLS closely followed MFRFNN with continuous states in the $y(t)$ and $z(t)$ series.

Table 10
Five-step-ahead prediction error comparison on the AQI dataset ($PM_{2.5}$, PM_{10}, and SO_2)

Method	$PM_{2.5}$			PM_{10}			SO_2		
	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value
DT	1.17E+00	6.58E−01	4.58E−63	1.13E+00	6.86E−01	3.18E−37	1.02E+00	6.34E−01	1.30E−74
	(1.30E−02)	(4.01E−03)		(1.41E−02)	(4.48E−03)		(5.59E−03)	(3.40E−03)	
RF	9.69E−01	5.35E−01	1.25E−45	9.00E−01	5.47E−01	7.84E−44	7.65E−01	4.70E−01	1.02E−50
	(4.10E−03)	(1.76E−03)		(5.05E−03)	(2.71E−03)		(1.53E−03)	(1.72E−03)	
SVR	9.85E−01	5.13E−01	8.21E−38	8.89E−01	5.04E−01	1.39E−67	7.31E−01	4.31E−01	9.69E−43
	(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)	
MLP	9.35E−01	5.03E−01	2.50E−37	8.87E−01	5.23E−01	1.46E−67	7.27E−01	4.32E−01	1.09E−42
	(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)	
LSTM	9.45E−01	5.13E−01	7.90E−62	8.99E−01	5.34E−01	8.83E−39	7.38E−01	4.39E−01	3.20E−42
	(9.22E−03)	(1.37E−02)		(9.13E−03)	(8.53E−03)		(1.22E−02)	(8.01E−03)	
EMD-LSTM	7.02E−01	4.02E−01	7.82E−50	8.84E−01	5.33E−01	3.75E−32	7.12E−01	4.28E−01	1.01E−44
	(1.50E−02)	(1.06E−02)		(1.99E−02)	(1.71E−02)		(1.09E−02)	(6.66E−03)	
WT-LSTM	8.32E−01	4.35E−01	6.24E−58	8.09E−01	4.81E−01	8.68E−29	6.48E−01	3.70E−01	2.45E−49
	(8.25E−03)	(1.34E−02)		(2.71E−02)	(2.25E−02)		(9.02E−03)	(8.24E−03)	
VMD-LSTM	3.69E−01	2.04E−01	1.33E−37	3.34E−01	2.04E−01	2.37E−23	3.17E−01	1.77E−01	4.48E−53
	(5.89E−03)	(9.57E−03)		(1.80E−02)	(1.04E−02)		(3.09E−03)	(2.74E−03)	
NLSTM	9.38E−01	4.99E−01	9.60E−55	8.84E−01	5.34E−01	3.24E−39	7.41E−01	4.47E−01	8.93E−39
	(6.55E−03)	(1.16E−02)		(8.51E−03)	(1.07E−02)		(1.43E−02)	(1.14E−02)	
EMD-NLSTM	7.59E−01	4.23E−01	2.43E−27	8.74E−01	5.15E−01	2.12E−33	7.13E−01	4.28E−01	7.45E−40
	(3.97E−02)	(2.07E−02)		(1.69E−02)	(1.68E−02)		(1.33E−02)	(8.79E−03)	
WT-NLSTM	8.35E−01	4.43E−01	3.67E−50	7.94E−01	4.59E−01	1.44E−35	6.45E−01	3.69E−01	1.65E−56
	(1.63E−02)	(1.71E−02)		(1.17E−02)	(1.11E−02)		(7.40E−03)	(6.82E−03)	
VMD-NLSTM	3.77E−01	2.06E−01	6.57E−35	3.23E−01	1.96E−01	1.24E−32	3.13E−01	1.76E−01	3.22E−56
	(1.69E−02)	(1.12E−02)		(5.65E−03)	(6.06E−03)		(3.78E−03)	(3.74E−03)	
SLSTM	9.56E−01	5.25E−01	6.03E−62	8.92E−01	5.29E−01	1.99E−36	7.33E−01	4.37E−01	1.15E−40
	(1.14E−02)	(2.66E−02)		(1.20E−02)	(1.41E−02)		(1.30E−02)	(8.38E−03)	
EMD-SLSTM	7.20E−01	3.90E−01	7.82E−28	8.58E−01	5.03E−01	2.41E−37	7.20E−01	4.27E−01	4.09E−44
	(3.69E−02)	(1.90E−02)		(1.03E−02)	(1.33E−02)		(1.12E−02)	(1.10E−02)	
WT-SLSTM	8.46E−01	4.61E−01	2.23E−48	8.21E−01	4.80E−01	4.46E−32	6.53E−01	3.78E−01	1.04E−38
	(1.73E−02)	(2.02E−02)		(1.85E−02)	(1.10E−02)		(1.34E−02)	(1.05E−02)	
VMD-SLSTM	3.47E−01	1.91E−01	4.78E−40	3.17E−01	1.96E−01	3.07E−22	3.04E−01	1.74E−01	3.08E−44
	(7.58E−03)	(7.94E−03)		(1.92E−02)	(1.31E−02)		(7.43E−03)	(6.90E−03)	
MTMC-NLSTM	6.43E−01	3.48E−01	7.73E−24	5.95E−01	3.40E−01	1.44E−25	5.20E−01	3.11E−01	3.77E−27
	(4.50E−02)	(2.80E−02)		(2.82E−02)	(2.52E−02)		(2.45E−02)	(1.75E−02)	
MFRFNN	**1.08E−01**	**4.90E−02**	–	**8.60E−02**	**5.17E−02**	–	**6.17E−02**	**3.26E−02**	–
	(1.10E−02)	**(1.53E−03)**		**(2.73E−04)**	**(2.41E−04)**		**(4.62E−03)**	**(4.36E−04)**	

Table 11
Five-step-ahead prediction error comparison on the AQI dataset (NO_2, CO, and O_3).

Method	NO_2			CO			O_3		
	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value
DT	1.13E+00 (5.21E−03)	8.34E−01 (2.83E−03)	7.82E−46	1.58E+00 (1.81E−02)	9.80E−01 (9.37E−03)	2.22E−48	7.93E−01 (9.61E−03)	5.39E−01 (4.67E−03)	3.44E−40
RF	8.51E−01 (2.71E−03)	6.60E−01 (2.58E−03)	3.21E−50	1.27E+00 (7.19E−03)	7.99E−01 (3.14E−03)	1.33E−72	5.19E−01 (4.38E−03)	3.94E−01 (3.15E−03)	4.60E−53
SVR	7.87E−01 (0.00E+00)	5.85E−01 (0.00E+00)	2.69E−62	1.40E+00 (0.00E+00)	8.09E−01 (0.00E+00)	1.72E−43	3.85E−01 (0.00E+00)	2.82E−01 (0.00E+00)	9.04E−44
MLP	7.97E−01 (0.00E+00)	6.14E−01 (0.00E+00)	2.04E−62	1.19E+00 (0.00E+00)	7.44E−01 (0.00E+00)	5.62E−42	4.31E−01 (0.00E+00)	3.52E−01 (0.00E+00)	7.75E−45
LSTM	8.09E−01 (1.60E−02)	6.26E−01 (2.00E−02)	5.54E−33	1.23E+00 (2.94E−02)	7.64E−01 (1.50E−02)	8.46E−35	4.47E−01 (2.91E−02)	3.62E−01 (2.54E−02)	2.22E−23
EMD-LSTM	6.62E−01 (2.22E−02)	5.02E−01 (2.66E−02)	2.44E−28	1.18E+00 (2.78E−02)	7.33E−01 (1.53E−02)	2.89E−35	3.45E−01 (2.78E−02)	2.59E−01 (3.05E−02)	2.67E−21
WT-LSTM	6.92E−01 (1.88E−02)	5.31E−01 (1.78E−02)	3.79E−30	1.09E+00 (3.50E−02)	6.68E−01 (1.95E−02)	5.45E−31	3.45E−01 (2.68E−02)	2.77E−01 (2.71E−02)	1.29E−21
VMD-LSTM	2.82E−01 (3.47E−02)	2.09E−01 (2.68E−02)	2.10E−15	5.16E−01 (3.61E−02)	3.02E−01 (1.41E−02)	3.97E−22	1.41E−01 (4.30E−02)	1.14E−01 (4.23E−02)	1.70E−08
NLSTM	8.06E−01 (1.39E−02)	6.21E−01 (2.05E−02)	4.01E−34	1.23E+00 (4.42E−02)	7.65E−01 (2.10E−02)	2.25E−29	4.40E−01 (1.51E−02)	3.57E−01 (1.74E−02)	3.11E−29
EMD-NLSTM	6.60E−01 (1.78E−02)	4.97E−01 (2.10E−02)	3.85E−30	1.17E+00 (2.14E−02)	7.21E−01 (1.04E−02)	2.64E−40	3.08E−01 (6.90E−03)	2.14E−01 (3.82E−03)	1.55E−35
WT-NLSTM	6.87E−01 (1.09E−02)	5.22E−01 (1.57E−02)	1.23E−34	1.09E+00 (7.37E−02)	6.72E−01 (4.02E−02)	4.45E−23	3.53E−01 (1.14E−02)	2.87E−01 (1.35E−02)	4.90E−30
VMD-NLSTM	2.62E−01 (1.63E−02)	1.97E−01 (1.31E−02)	1.38E−20	5.62E−01 (3.33E−02)	3.17E−01 (1.69E−02)	3.81E−24	1.41E−01 (4.69E−02)	1.14E−01 (4.61E−02)	6.57E−08
SLSTM	8.24E−01 (3.01E−02)	6.44E−01 (3.71E−02)	6.46E−28	1.25E+00 (5.46E−02)	7.80E−01 (3.92E−02)	2.83E−27	4.25E−01 (8.67E−03)	3.38E−01 (1.37E−02)	1.90E−35
EMD-SLSTM	6.28E−01 (2.30E−02)	4.63E−01 (2.86E−02)	1.58E−27	1.20E+00 (3.82E−02)	7.50E−01 (2.08E−02)	7.37E−31	3.43E−01 (8.57E−02)	2.45E−01 (8.07E−02)	4.32E−12
WT-SLSTM	6.99E−01 (1.27E−02)	5.41E−01 (1.88E−02)	1.62E−33	1.08E+00 (4.83E−02)	6.60E−01 (3.02E−02)	4.22E−27	3.53E−01 (1.97E−02)	2.87E−01 (2.28E−02)	1.48E−24
VMD-SLSTM	2.52E−01 (2.39E−02)	1.93E−01 (2.19E−02)	6.15E−17	4.98E−01 (3.07E−02)	2.95E−01 (1.44E−02)	1.27E−23	1.10E−01 (6.73E−03)	8.43E−02 (6.99E−03)	1.30E−21
MTMC-NLSTM	5.88E−01 (2.96E−02)	4.29E−01 (2.33E−02)	8.55E−25	8.41E−01 (5.23E−02)	4.89E−01 (2.55E−02)	9.98E−24	3.51E−01 (2.25E−02)	2.56E−01 (1.85E−02)	2.62E−23
MFRFNN	**1.02E−01 (4.42E−04)**	**7.82E−02 (4.67E−04)**	–	**1.48E−01 (7.89E−03)**	**9.17E−02 (3.40E−03)**	–	**5.18E−02 (2.03E−03)**	**3.81E−02 (1.77E−03)**	–

Although MFRFNN with continuous states outperformed the RBLS in all evaluation metrics, its standard deviation was higher than RBLS. A higher standard deviation means a higher variance of the predictions, which is the drawback of MFRFNN. The results obtained by RBLS in the $y(t)$ and $z(t)$ series were closely followed by ESN. The good performance of RBLS is due to the recurrently connected nodes in its enhancement units, allowing it to capture the dynamic behavior of the Lorenz time series. As can be seen from Table 4, the $\varepsilon-$ SVR method had the worst performance on all metrics, and its standard deviation was zero because there was no randomness in this algorithm. Fig. 4 shows that the maximum errors for the $x(t), y(t)$, and $z(t)$ series do not exceed $2\times10^{-4}, 4.5\times10^{-3}$, and 7×10^{-3}, respectively, which proves that the results obtained by MFRFNN with continuous states were acceptable. Note that the prediction intervals for the mentioned series were $[-20, 20]$, $[-30, 30]$, and $[0, 50]$, respectively. Moreover, from histograms of errors, it can be seen that the absolute errors followed a normal distribution, meaning that MFRFNN effectively captured the dynamic characteristics of the Lorenz system due to its capability of learning multiple functions simultaneously.

In Rossler time series experiments, the results showed better performance of MFRFNN with continuous states for the same prediction horizon compared to other methods. After MFRFNN with continuous states, MFRFNN with discrete states had the lowest RMSE, closely followed by HESN. We also noted that L_1ESN and L_2ESN had the worst accuracy in most cases. As can be seen from Fig. 5, MFRFNN with continuous states had the lowest RMSE in various prediction horizons. When the prediction horizon increased from 1 to 23, the RMSE difference between MFRFNN with continuous states and HESN increased from 0.005 to 0.506. For the prediction horizon of 24 to 28, the RMSE difference was 0.470 on average. The long-term prediction ability of the proposed method can be explained by its structure. As mentioned, having a structure with a feedback loop makes the model capable of memorizing historical information. Furthermore, using multiple states helps the proposed method store past observations and track system dynamics in larger prediction horizons.

The results of the Box–Jenkins gas furnace problem experiments showed that MFRFNN with continuous states outperformed other methods in this dataset, closely followed by MFRFNN with discrete states and global BaNFIS. Based on RMSE, MFRFNN with continuous states showed a decrease of $19.57\%, 26.00\%, 13.95\%$, and 41.27% from the NFIS-DN, SPATFIS, global BaNFIS, and local BaNFIS, respectively. Based on MSE, MFRFNN with continuous states showed a decrease of $50.00\%, 66.67\%, 50.00\%$, and 75.00% from the NFIS-DN, SPATFIS, global BaNFIS, and local BaNFIS, respectively. Based on MAE, MFRFNN with continuous states showed a decrease of $21.21\%, 27.78\%, 10.34\%$, and 46.94% from the NFIS-DN, SPATFIS, global BaNFIS, and local BaNFIS, respectively. McFIS had the highest accuracy among FNNs. Roughly speaking, RFNNs obtained better results compared to FNNs. Since FNNs do not have recurrent connections, they could not learn temporal dependencies in this prediction task. Box–Jenkins time series exhibit serial autocorrelation, and by plotting the partial autocorrelation function, it can be seen that the first four lags are statistically significant [76]. So, the future value depends on four past observations. Therefore, the good performance of MFRFNN in this dataset can be explained by its structure. Having a structure with a feedback loop makes the

Table 12
Ten-step-ahead prediction error comparison on the AQI dataset ($PM_{2.5}$, PM_{10}, and SO_2).

Method	$PM_{2.5}$			PM_{10}			SO_2		
	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value
DT	1.43E+00 (7.22E−03)	8.76E−01 (4.65E−03)	4.99E−52	1.39E+00 (4.99E−03)	8.96E−01 (5.08E−03)	9.65E−48	1.17E+00 (9.29E−03)	7.32E−01 (5.43E−03)	1.64E−43
RF	1.21E+00 (2.57E−03)	7.28E−01 (8.08E−04)	1.80E−90	1.14E+00 (5.17E−03)	7.23E−01 (3.18E−03)	1.25E−45	8.88E−01 (1.90E−03)	5.68E−01 (1.11E−03)	6.16E−91
SVR	1.20E+00 (0.00E+00)	6.80E−01 (0.00E+00)	1.84E−52	1.12E+00 (0.00E+00)	6.68E−01 (0.00E+00)	4.96E−69	8.10E−01 (0.00E+00)	5.02E−01 (0.00E+00)	1.36E−51
MLP	1.17E+00 (0.00E+00)	6.93E−01 (0.00E+00)	3.14E−52	1.11E+00 (0.00E+00)	6.97E−01 (0.00E+00)	5.99E−69	8.41E−01 (0.00E+00)	5.18E−01 (0.00E+00)	6.27E−52
LSTM	1.17E+00 (8.20E−03)	7.02E−01 (1.54E−02)	4.23E−47	1.11E+00 (3.01E−02)	6.99E−01 (1.87E−02)	1.28E−30	8.24E−01 (1.03E−02)	5.25E−01 (6.03E−03)	6.87E−39
EMD-LSTM	1.17E+00 (1.48E−02)	7.02E−01 (1.81E−02)	2.49E−38	1.06E+00 (5.43E−03)	6.89E−01 (1.40E−02)	1.57E−44	8.20E−01 (8.13E−03)	5.17E−01 (5.06E−03)	5.30E−42
WT-LSTM	1.14E+00 (2.31E−02)	6.73E−01 (1.34E−02)	1.67E−33	1.07E+00 (1.39E−02)	6.84E−01 (1.13E−02)	1.12E−36	8.01E−01 (6.04E−03)	5.10E−01 (5.15E−03)	6.32E−47
VMD-LSTM	6.05E−01 (1.04E−02)	3.59E−01 (2.27E−02)	4.90E−36	5.73E−01 (2.65E−02)	3.47E−01 (7.02E−03)	2.35E−25	4.55E−01 (1.16E−02)	2.76E−01 (8.08E−03)	9.19E−32
NLSTM	1.17E+00 (7.73E−03)	6.98E−01 (1.52E−02)	2.56E−48	1.10E+00 (1.63E−02)	7.15E−01 (1.75E−02)	1.31E−35	8.23E−01 (1.40E−02)	5.21E−01 (1.37E−02)	1.76E−35
EMD-NLSTM	1.14E+00 (1.91E−02)	6.74E−01 (1.66E−02)	2.36E−35	1.08E+00 (8.30E−03)	6.85E−01 (1.17E−02)	4.49E−41	8.18E−01 (1.06E−02)	5.21E−01 (6.64E−03)	1.78E−38
WT-NLSTM	1.14E+00 (1.84E−02)	6.77E−01 (1.73E−02)	9.89E−36	1.07E+00 (4.34E−03)	6.77E−01 (1.23E−02)	1.33E−46	8.10E−01 (6.37E−03)	5.07E−01 (2.03E−03)	5.51E−46
VMD-NLSTM	6.06E−01 (2.99E−02)	3.52E−01 (2.13E−02)	4.99E−25	5.73E−01 (1.41E−02)	3.61E−01 (2.25E−02)	1.44E−30	4.50E−01 (5.52E−03)	2.71E−01 (4.21E−03)	1.83E−42
SLSTM	1.17E+00 (1.20E−02)	7.01E−01 (2.17E−02)	6.36E−41	1.09E+00 (8.20E−03)	6.95E−01 (1.54E−02)	2.92E−41	8.20E−01 (1.34E−02)	5.24E−01 (8.49E−03)	6.73E−36
EMD-SLSTM	1.13E+00 (1.64E−02)	6.71E−01 (2.19E−02)	7.43E−37	1.08E+00 (1.03E−02)	6.91E−01 (1.61E−02)	2.91E−39	8.21E−01 (1.08E−02)	5.22E−01 (7.47E−03)	2.74E−38
WT-SLSTM	1.14E+00 (1.67E−02)	6.70E−01 (1.12E−02)	9.59E−37	1.07E+00 (7.46E−03)	6.73E−01 (7.44E−03)	6.84E−42	7.99E−01 (8.06E−03)	5.06E−01 (6.02E−03)	7.08E−42
VMD-SLSTM	5.65E−01 (1.56E−02)	3.23E−01 (6.24E−03)	2.01E−30	5.57E−01 (3.82E−02)	3.41E−01 (1.90E−02)	4.62E−22	4.44E−01 (9.76E−03)	2.65E−01 (7.57E−03)	1.84E−33
MTMC-NLSTM	8.71E−01 (5.74E−02)	4.90E−01 (3.45E−02)	4.78E−23	8.34E−01 (4.40E−02)	4.82E−01 (2.72E−02)	7.70E−25	5.15E−01 (2.29E−02)	3.38E−01 (3.27E−02)	1.70E−26
MFRFNN	**1.13E−01 (2.31E−03)**	**6.44E−02 (3.88E−04)**	–	**1.07E−01 (2.89E−04)**	**6.77E−02 (2.65E−04)**	–	**6.45E−02 (1.76E−03)**	**3.91E−02 (3.84E−04)**	–

model capable of memorizing past observations and using them to predict the future value. Note that as serial autocorrelation is present in this dataset, it requires more past observation in its input space than in its latent space. As a result, global BaNFIS outperformed local BaNFIS in this experiment.

In the wind speed prediction dataset, MFRFNN with continuous states obtained the smallest RMSE, MSE, and MAE. Based on RMSE, MFRFNN with continuous states showed a decrease of $55.33\%, 54.11\%, 50.74\%$, and 49.62% from the NFIS-DN, SPATFIS, global BaNFIS, and local BaNFIS, respectively. Based on MSE, MFRFNN with continuous states showed a decrease of $77.27\%, 76.19\%, 72.22\%$, and 70.59% from the NFIS-DN, SPATFIS, global BaNFIS, and local BaNFIS, respectively. Based on MAE, MFRFNN with continuous states showed a decrease of $54.72\%, 52.48\%, 51.52\%$, and 50.52% from the NFIS-DN, SPATFIS, global BaNFIS, and local BaNFIS, respectively. GENEFIS was the most accurate method among FNNs, and RFNNs outperformed FNNs in all evaluation metrics. It is worth noting that local BaNFIS outperformed global BaNFIS in this dataset. As aforementioned, the wind speed dataset is more challenging than the Box–Jenkins dataset, so a method based on an autoregressive model is not capable of capturing the dynamic characteristics of this time series. Moreover, we noted that although the eTS results on the Box–Jenkins dataset were satisfying, it had the worst performance on all metrics for the wind speed prediction dataset.

The results of Google stock price prediction indicated that global BaNFIS outperformed other methods, closely followed by MFRFNN with continuous and discrete states. Based on RMSE, global BaNFIS showed a decrease of $46.67\%, 20.00\%$, and 5.88% from the NFIS-DN, SPATFIS, and MFRFNN, respectively. The good performance of global BaNFIS in this dataset was due to its capability to handle dynamics and use only required past observations in its prediction. Same as wind speed prediction, GENEFIS outperformed other FNNs in this experiment.

In five-step-ahead and ten-step-ahead prediction experiments on the AQI dataset, MFRFNN achieved the best results in all components, closely followed by VMD-SLSTM. When the prediction horizon increased from 5 to 10, the RMSE difference between MFRFNN and the second best method in each component increased too. The increase in this difference varied from component to component, rising from 0.239 to 0.425 for $PM_{2.5}$, from 0.231 to 0.450 for PM_{10}, from 0.242 to 0.380 for SO_2, from 0.150 to 0.482 for NO_2, from 0.350 to 0.646 for CO, and from 0.058 to 0.184 for O_3. The good performance of MFRFNN in this dataset can be attributed to the superiority of its structure and the feedback loop between the output network and state network. Using two states in these experiments makes MFRFNN capable of learning and tracking various behaviors of the AQI time series with two separate functions (i.e., a single function for each state). Moreover, comparing the same methods with and without decomposition showed a positive impact of decomposition techniques on prediction results. Decomposition methods such as EMD and VMD decompose time series into several intrinsic mode functions with a simpler structure and reduce the complexity of time series. As a result, it could be predicted more easily than a high fluctuating time series. Methods using VMD (i.e., VMD-NLSTM and VMD-LSTM) exhibited superior performance over EMD-based methods. EMD's inherent problems can explain the worse performance of EMD-based methods. EMD

Table 13
Ten-step-ahead prediction error comparison on the AQI dataset (NO_2, CO, and O_3).

Method	NO_2			CO			O_3		
	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value	RMSE	MAE	*p*-value
DT	1.37E+00	1.06E+00	2.22E−51	1.70E+00	1.10E+00	1.05E−61	1.07E+00	7.35E−01	9.09E−61
	(4.51E−03)	(4.79E−03)		(8.86E−03)	(1.05E−02)		(8.27E−03)	(6.43E−03)	
RF	1.06E+00	8.48E−01	5.77E−61	1.49E+00	9.78E−01	8.30E−78	6.55E−01	5.27E−01	1.39E−69
	(2.56E−03)	(1.96E−03)		(5.86E−03)	(4.71E−03)		(4.96E−03)	(3.35E−03)	
SVR	1.02E+00	7.91E−01	1.80E−58	1.61E+00	9.82E−01	5.20E−49	4.19E−01	3.39E−01	4.79E−37
	(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)	
MLP	1.00E+00	8.06E−01	2.76E−58	1.54E+00	9.69E−01	1.35E−48	5.30E−01	4.54E−01	2.66E−39
	(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)		(0.00E+00)	(0.00E+00)	
LSTM	1.02E+00	8.26E−01	2.80E−35	1.58E+00	9.87E−01	4.69E−35	5.66E−01	4.84E−01	1.15E−24
	(1.56E−02)	(1.76E−02)		(2.88E−02)	(1.81E−02)		(3.36E−02)	(2.74E−02)	
EMD-LSTM	9.88E−01	7.99E−01	1.15E−40	1.53E+00	9.65E−01	8.55E−36	5.08E−01	4.15E−01	1.29E−24
	(8.40E−03)	(1.07E−02)		(2.63E−02)	(1.43E−02)		(3.04E−02)	(3.13E−02)	
WT-LSTM	9.82E−01	7.91E−01	1.52E−33	1.51E+00	9.48E−01	3.89E−26	5.84E−01	5.00E−01	3.42E−29
	(1.83E−02)	(1.56E−02)		(7.00E−02)	(3.29E−02)		(2.29E−02)	(1.83E−02)	
VMD-LSTM	6.78E−01	5.27E−01	6.79E−23	8.31E−01	4.99E−01	7.48E−21	2.58E−01	2.02E−01	3.55E−22
	(4.24E−02)	(4.50E−02)		(6.49E−02)	(2.95E−02)		(1.95E−02)	(2.33E−02)	
NLSTM	1.01E+00	8.11E−01	6.95E−36	1.54E+00	9.73E−01	1.95E−31	5.59E−01	4.80E−01	6.58E−26
	(1.44E−02)	(2.18E−02)		(3.98E−02)	(2.17E−02)		(2.94E−02)	(2.24E−02)	
EMD-NLSTM	9.95E−01	8.02E−01	6.53E−33	1.55E+00	9.71E−01	7.56E−32	4.95E−01	3.85E−01	4.11E−37
	(2.00E−02)	(2.02E−02)		(3.84E−02)	(1.63E−02)		(1.27E−02)	(1.61E−02)	
WT-NLSTM	9.81E−01	7.91E−01	1.51E−41	1.47E+00	9.33E−01	2.94E−38	5.56E−01	4.76E−01	4.14E−27
	(7.65E−03)	(7.33E−03)		(2.11E−02)	(1.22E−02)		(2.62E−02)	(2.29E−02)	
VMD-NLSTM	6.75E−01	5.33E−01	6.48E−22	8.53E−01	5.14E−01	2.58E−21	2.75E−01	2.13E−01	6.24E−20
	(4.75E−02)	(5.39E−02)		(6.35E−02)	(2.99E−02)		(2.55E−02)	(2.45E−02)	
SLSTM	1.02E+00	8.20E−01	1.55E−34	1.53E+00	9.68E−01	3.38E−29	5.43E−01	4.66E−01	7.16E−34
	(1.70E−02)	(1.80E−02)		(5.02E−02)	(2.39E−02)		(1.58E−02)	(1.29E−02)	
EMD-SLSTM	9.71E−01	7.68E−01	1.13E−33	1.51E+00	9.51E−01	2.78E−28	5.45E−01	4.26E−01	1.27E−25
	(1.78E−02)	(1.79E−02)		(5.48E−02)	(2.81E−02)		(2.95E−02)	(2.45E−02)	
WT-SLSTM	9.79E−01	7.86E−01	2.74E−36	1.47E+00	9.34E−01	5.98E−32	5.05E−01	4.29E−01	1.26E−26
	(1.33E−02)	(1.63E−02)		(3.61E−02)	(1.95E−02)		(2.51E−02)	(2.18E−02)	
VMD-SLSTM	6.08E−01	4.78E−01	4.71E−24	8.28E−01	4.98E−01	1.45E−24	2.50E−01	1.87E−01	1.33E−38
	(3.22E−02)	(3.59E−02)		(4.24E−02)	(2.61E−02)		(8.14E−03)	(9.44E−03)	
MTMC-NLSTM	8.01E−01	6.33E−01	6.29E−25	1.18E+00	6.86E−01	2.87E−23	5.22E−01	4.27E−01	2.00E−22
	(4.05E−02)	(3.23E−02)		(7.42E−02)	(4.06E−02)		(3.91E−02)	(3.29E−02)	
MFRFNN	**1.26E−01**	**1.01E−01**	–	**1.82E−01**	**1.17E−01**	–	**6.59E−02**	**4.85E−02**	–
	(9.17E−04)	**(6.89E−04)**		**(4.61E−03)**	**(2.17E−03)**		**(4.86E−03)**	**(2.88E−03)**	

Table 14
Comparison of the number of parameters and average training time.

Method	Number of Parameters	Average Training Time (s)
LSTM	42,197	163.7
EMD-LSTM	42,197	453.4
WT-LSTM	42,197	677.53
VMD-LSTM	50,977	650.47
NLSTM	50,977	166.86
EMD-NLSTM	50,977	451.81
WT-NLSTM	50,977	672.71
VMD-NLSTM	50,977	682.59
SLSTM	182,081	378.42
EMD-SLSTM	182,081	1101.78
WT-SLSTM	182,081	1777.79
VMD-SLSTM	182,081	1891.42
MTMC-NLSTM	345,030	211.91
MFRFNN	64	236.74

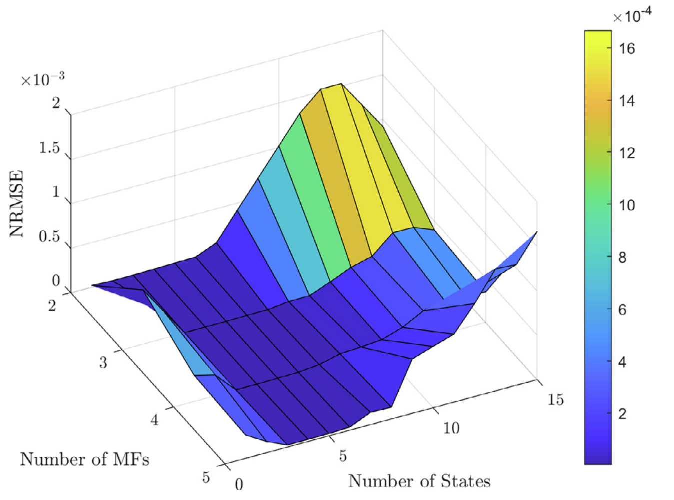


Fig. 6. Relationship between NRMSE, the number of MFs, and the number of states of the one-step-ahead prediction of the Lorenz time series.

has drawbacks such as endpoint effect, mode-mixing, sensitivity to noise, and the need to choose a specific function for interpolation. Regarding the number of parameters in this experiment, MFRFNN had the lowest number of parameters. The proposed network used two states and 16 rules for each network, so it had 64 parameters ($2 \times 2 \times 16$). Note that MFRFNN had the lowest number of parameters but not the lowest average training time. MFRFNN required calculating fuzzy membership values in each iteration of the training algorithm, which increased its computational cost. The LSTM had the lowest average training time. The average training time increased with the increase of the number of parameters. MTMC-NLSTM had the highest number of parameters but not the highest average training time because it simultaneously created the prediction model for different AQI components. From Table 14, it is evident that pre-processing methods increase the training time drastically, and VMD-SLSTM had the highest average training time.

Finally, the results of comparing different metaheuristic algorithms indicated that based on RMSE, the PSO algorithm outperformed other methods in the training of MFRFNN. Moreover, BA, IWO, and PSO obtained the smallest MAE. The standard deviation

Table 15
NRMSE for one-step-ahead prediction of the Box–Jenkins gas furnace problem.

#States/#MFs	2	3	4	5
1	5.10E−02	5.23E−02	5.10E−02	5.25E−02
2	3.34E−02	3.58E−02	3.58E−02	4.05E−02
3	**3.04E−02**	3.44E−02	3.87E−02	4.30E−02
4	3.51E−02	3.51E−02	4.41E−02	5.05E−02
5	3.64E−02	4.38E−02	4.75E−02	6.24E−02
6	3.88E−02	4.49E−02	5.25E−02	8.61E−02
7	4.54E−02	5.10E−02	6.73E−02	9.16E−02
8	4.30E−02	5.00E−02	8.18E−02	1.42E−01
9	4.66E−02	5.33E−02	9.08E−02	2.24E−01
10	4.59E−02	5.25E−02	1.34E−01	2.61E−01
11	4.61E−02	5.91E−02	1.61E−01	2.83E−01
12	4.77E−02	5.14E−02	1.90E−01	5.18E−01
13	4.81E−02	7.23E−02	2.86E−01	4.01E−01
14	4.24E−02	8.23E−02	3.41E−01	6.38E−01
15	5.13E−02	8.54E−02	3.65E−01	4.07E−01

of PSO was lower than other metaheuristic algorithms. A lower standard deviation means a lower variance of predictions and a more robust training algorithm. The good performance of the PSO algorithm, and the key reason this method was used for training of MFRFNN, comes with its ability to handle optimization problems with multiple local optima reasonably well and its global search capability. CMA-ES and FA obtained the lowest and highest training time, respectively.

Table 18
Ten-step-ahead prediction error comparison of different metaheuristic algorithms on AQI dataset (SO_2).

Method	RMSE	MAE	Training Time (s)	*p*-value
ABC	7.86E−02 (1.16E−02)	3.93E−02 (4.96E−04)	235.47	1.43E−04
ACOR	7.41E−02 (1.05E−02)	4.00E−02 (7.98E−04)	238.41	6.48E−04
BA	8.04E−02 (2.62E−02)	**3.91E−02 (9.07E−04)**	228.81	1.39E−02
BBO	8.06E−02 (1.85E−02)	3.94E−02 (8.04E−04)	236.16	9.93E−04
CMA-ES	7.68E−02 (1.28E−02)	3.92E−02 (5.59E−04)	**219.66**	3.96E−04
DE	7.27E−02 (1.33E−02)	3.95E−02 (9.08E−04)	238.02	1.29E−02
FA	7.84E−02 (2.00E−02)	3.98E−02 (9.36E−04)	246.07	5.87E−03
GA	8.30E−02 (2.36E−02)	3.95E−02 (8.31E−04)	237.14	2.39E−03
HS	7.36E−02 (1.05E−02)	3.94E−02 (5.33E−04)	237.93	1.06E−03
ICA	7.80E−02 (1.39E−02)	3.92E−02 (7.79E−04)	241.08	3.55E−04
IWO	7.86E−02 (2.97E−02)	**3.91E−02 (1.11E−03)**	235.99	4.74E−02
TLBO	7.90E−02 (1.74E−02)	3.96E−02 (7.76E−04)	239.18	1.45E−03
MFRFNN (PSO)	**6.45E−02 (1.76E−03)**	**3.91E−02 (3.84E−04)**	236.11	–

7. Conclusion

This paper proposed a novel multi-functional recurrent fuzzy neural network for chaotic time series prediction. MFRFNN consisted of two FNNs with TSK fuzzy rules, one was used to produce the system's output, and the other to determine the system's state. The feedback loop between these two networks made MFRFNN capable of learning and memorizing historical information of past observations. Employing the states allowed the proposed network to learn multiple functions simultaneously, resulting in capturing the dynamic behavior of the chaotic time series and predicting long-term values of the time series. Moreover, a new learning algorithm, which employed the PSO algorithm, was developed to train the weights of MFRFNN. The experimental results indicated that, for the Lorenz time series, based on the RMSE, MFRFNN showed a 35.12% decrease from the second best method (i.e., RBLS) on average. In the Rossler time series, when the prediction horizon increased from 1 to 23, the RMSE difference between MFRFNN and the second best method (i.e., HESN) increased from 0.005 to 0.506. For the Box–Jenkins gas furnace dataset and wind speed prediction dataset, based on the RMSE, the proposed network showed

Table 16
Comparison of MFRFNN's performance with and without a feedback loop (Lorenz and Rossler).

Benchmark	With Feedback Loop		Without Feedback Loop		*p*-value
	RMSE	SMAPE	RMSE	SMAPE	
Lorenz System $x(t)$	**2.44E−05**	**7.27E−07**	1.53E−04	8.01E−06	3.60E−26
Lorenz System $y(t)$	**3.58E−04**	**1.40E−05**	4.87E−03	2.38E−04	8.86E−41
Lorenz System $z(t)$	**4.36E−04**	**4.79E−06**	2.22E−03	2.90E−05	2.40E−31
Rossler System $x(t)$	**6.22E−07**	**1.15E−08**	2.78E−06	7.30E−08	2.56E−16
Rossler System $y(t)$	**1.49E−09**	**2.92E−11**	6.99E−09	1.84E−10	1.55E−17
Rossler System $z(t)$	**1.47E−04**	**2.47E−05**	8.29E−04	1.57E−04	8.55E−22

Table 17
Comparison of MFRFNN's performance with and without a feedback loop (Real-world datasets).

Benchmark	With Feedback Loop		Without Feedback Loop		p-value
	RMSE	MAE	RMSE	MAE	
AQI-$PM_{2.5}$	**1.13E−01**	**6.44E−02**	1.18E−01	6.69E−02	8.86E−09
AQI-PM_{10}	1.07E−01	**6.77E−02**	1.07E−01	6.92E−02	1.00E+00
AQI-SO_2	**6.45E−02**	3.91E−02	6.80E−02	**3.79E−02**	3.36E−08
AQI-NO_2	**1.26E−01**	**1.01E−01**	1.31E−01	1.06E−01	8.44E−16
AQI-CO	**1.82E−01**	**1.17E−01**	1.98E−01	1.23E−01	3.01E−12
AQI-O_3	**6.59E−02**	**4.85E−02**	7.34E−02	6.29E−02	1.40E−06
Google Stock	**1.73E−02**	**1.20E−02**	1.22E−01	7.37E−02	1.40E−47
Box–Jenkins	**3.68E−02**	**2.60E−02**	5.23E−02	4.07E−02	4.27E−21
Wind Speed	**6.72E−02**	**4.77E−02**	7.03E−02	5.29E−02	3.18E−12

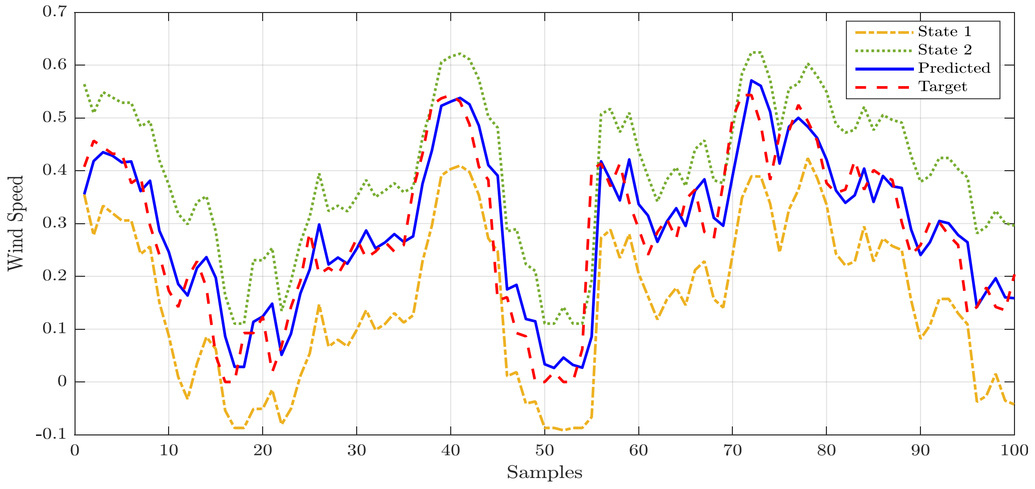


Fig. 7. Functions learned by each state, one-step-ahead prediction curve, and target curve for the wind speed prediction dataset.

a decrease of 13.95% and 49.62% from the second best method, respectively. In the Google stock price prediction task, MFRFNN was the second best method after the global BaNFIS. In the AQI dataset it showed promising results and outperformed other methods in five-step-ahead and ten-step-ahead prediction tasks. Overall, the experimental results showed that MFRFNN outperformed other state-of-the-art methods for both chaotic benchmarks and real-world datasets and showed a good performance in the long-term prediction of the Rossler system and AQI dataset. In future work, we would like to incorporate time series decomposition methods into MFRFNN.

Code availability

The source code of MFRFNN required to reproduce the predictions and results is available at the public Github repository[3].

CRediT authorship contribution statement

Hamid Nasiri: Methodology, Software, Validation, Writing - original draft. **Mohammad Mehdi Ebadzadeh:** Conceptualization, Investigation, Supervision.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] M. Han, K. Zhong, T. Qiu, B. Han, Interval type-2 fuzzy neural networks for chaotic time series prediction: A concise overview, IEEE Trans. Cybern. 49 (7) (2019) 2720–2731.
[2] C. Li, Z. Li, L. Guan, P. Qi, J. Si, B. Hao, Measuring the Complexity of Chaotic Time Series by Fuzzy Entropy, in, in: Proceedings of the International Conference on Future Networks and Distributed Systems, 2017, pp. 1–7.
[3] M. Xu, M. Han, C.L.P. Chen, T. Qiu, Recurrent Broad Learning Systems for Time Series Prediction, IEEE Trans. Cybern. 50 (4) (2020) 1405–1417, https://doi.org/10.1109/TCYB.2018.2863020.
[4] C.-H. Lee, F.-Y. Chang, C.-M. Lin, An efficient interval type-2 fuzzy CMAC for chaos time-series prediction and synchronization, IEEE Trans. Cybern. 44 (3) (2014) 329–341.
[5] M. Xu, M. Han, T. Qiu, H. Lin, Hybrid Regularized Echo State Network for Multivariate Chaotic Time Series Prediction, IEEE Trans. Cybern. 49 (6) (2019) 2305–2315.
[6] R. Castro, Y.M. Souto, E. Ogasawara, F. Porto, E. Bezerra, STconvS2S: Spatiotemporal convolutional sequence to sequence network for weather forecasting, Neurocomputing 426 (2021) 285–298.
[7] M.O. Alassafi, M. Jarrah, R. Alotaibi, Time series predicting of COVID-19 based on deep learning, Neurocomputing 468 (2022) 335–344.
[8] R. d. A. Araújo, N. Nedjah, A.L.I. Oliveira, R. d. L. Silvio, A deep increasing–decreasing-linear neural network for financial time series prediction, Neurocomputing 347 (2019) 59–81.
[9] M. Gan, C.L. Philip Chen, L. Chen, C.-Y. Zhang, Exploiting the interpretability and forecasting ability of the RBF-AR model for nonlinear time series, Int. J. Syst. Sci. 47 (8) (2016) 1868–1876.
[10] E. Hadavandi, A. Ghanbari, S. Abbasian-Naghneh, Developing a time series model based on particle swarm optimization for gold price forecasting, in: 2010 Third International Conference on Business Intelligence and Financial Engineering IEEE, 2010, pp. 337–340.
[11] M. Xu, M. Han, Adaptive elastic echo state network for multivariate time series prediction, IEEE Trans. Cybern. 46 (10) (2016) 2173–2183.
[12] H.-G. Han, Z.-L. Lin, J.-F. Qiao, Modeling of nonlinear systems using the self-organizing fuzzy neural network with adaptive gradient algorithm, Neurocomputing 266 (2017) 566–578.
[13] O. Khayat, M.M. Ebadzadeh, H.R. Shahdoosti, R. Rajaei, I. Khajehnasiri, A novel hybrid algorithm for creating self-organizing fuzzy neural networks, Neurocomputing 73 (1–3) (2009) 517–524.
[14] G. Khodabandelou, M.M. Ebadzadeh, Fuzzy neural network with support vector-based learning for classification and regression, Soft. Comput. 23 (23) (2019) 12153–12168.
[15] A. Salimi-Badr, M.M. Ebadzadeh, A novel learning algorithm based on computing the rules' desired outputs of a TSK fuzzy neural network with non-separable fuzzy rules, Neurocomputing 470 (2022) 139–153.
[16] T. Xie, F. Cao, The errors in simultaneous approximation by feed-forward neural networks, Neurocomputing 73 (4–6) (2010) 903–907.
[17] J. Qiao, F. Li, H. Han, W. Li, Growing Echo-State Network With Multiple Subreservoirs, IEEE Trans. Neural Networks Learn. Syst. 28 (2) (2017) 391–404.
[18] Q. Ma, S. Li, L. Shen, J. Wang, J. Wei, Z. Yu, G.W. Cottrell, End-to-end incomplete time-series modeling from linear memory of latent variables, IEEE Trans. Cybern. 50 (12) (2019) 4908–4920.
[19] Z. Li, G. Tanaka, Multi-reservoir echo state networks with sequence resampling for nonlinear time-series prediction, Neurocomputing 467 (2022) 115–129.
[20] X. Gong, T. Zhang, C.L.P. Chen, Z. Liu, Research Review for Broad Learning System: Algorithms, Theory, and Applications, IEEE Trans. Cybern.
[21] H. Wang, G. Song, Innovative NARX recurrent neural network model for ultra-thin shape memory alloy wire, Neurocomputing 134 (2014) 289–295.
[22] M. Ragab, Z. Chen, M. Wu, C.-K. Kwoh, R. Yan, X. Li, Attention-based sequence to sequence model for machine remaining useful life prediction, Neurocomputing 466 (2021) 58–68.
[23] H.-G. Han, Z.-Y. Chen, H.-X. Liu, J.-F. Qiao, A self-organizing interval Type-2 fuzzy-neural-network for modeling nonlinear systems, Neurocomputing 290 (2018) 196–207.
[24] C.-F. Juang, Y.-Y. Lin, C.-C. Tu, A recurrent self-evolving fuzzy neural network with local feedbacks and its application to dynamic system processing, Fuzzy Sets Syst. 161 (19) (2010) 2552–2568.
[25] Y.-Y. Lin, J.-Y. Chang, N.R. Pal, C.-T. Lin, A mutually recurrent interval type-2 neural fuzzy system (MRIT2NFS) with self-evolving structure and parameters, IEEE Trans. Fuzzy Syst. 21 (3) (2013) 492–509.
[26] S. Samanta, M. Pratama, S. Sundaram, A novel spatio-temporal fuzzy inference system (spatfis) and its stability analysis, Inf. Sci. 505 (2019) 84–99.
[27] S. Samanta, S. Suresh, J. Senthilnath, N. Sundararajan, A new neuro-fuzzy inference system with dynamic neurons (nfis-dn) for system identification and time series forecasting, Appl. Soft Comput. 82 (2019) 105567.

[3] https://github.com/Hamid-Nasiri/Recurrent-Fuzzy-Neural-Network.

[28] C. Luo, C. Tan, X. Wang, Y. Zheng, An evolving recurrent interval type-2 intuitionistic fuzzy neural network for online learning and time series prediction, Appl. Soft Comput. 78 (2019) 150–163.
[29] H. Ding, W. Li, J. Qiao, A self-organizing recurrent fuzzy neural network based on multivariate time series analysis, Neural Comput. Appl. 33 (10) (2021) 5089–5109.
[30] S. Subhrajit, P. Mahardhika, S. Sundaram, Bayesian Neuro-Fuzzy Inference System (BaNFIS) for Temporal Dependency Estimation, IEEE Trans. Fuzzy Syst. 29 (9) (2021) 2479–2490.
[31] R. Chandra, M. Zhang, Cooperative coevolution of Elman recurrent neural networks for chaotic time series prediction, Neurocomputing 86 (2012) 116–123.
[32] D. Li, X. Wang, J. Sun, Y. Feng, Radial basis function neural network model for dissolved oxygen concentration prediction based on an enhanced clustering algorithm and Adam, IEEE Access 9 (2021) 44521–44533.
[33] M. Zhu, Z. Meng, Macroeconomic Image Analysis and GDP Prediction Based on the Genetic Algorithm Radial Basis Function Neural Network (RBFNN-GA), Comput. Intell. Neurosci. (2021).
[34] H.-G. Han, M.-L. Ma, H.-Y. Yang, J.-F. Qiao, Self-organizing radial basis function neural network using accelerated second-order learning algorithm, Neurocomputing 469 (2022) 1–12.
[35] H. Jaeger, H. Haas, Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication, Science 304 (5667) (2004) 78–80.
[36] S. Scardapane, M. Panella, D. Comminiello, A. Hussain, A. Uncini, Distributed reservoir computing with sparse readouts [research frontier], IEEE Computat. Intell. Mag. 11 (4) (2016) 59–70.
[37] M. Han, W.-J. Ren, M.-L. Xu, An improved echo state network via l1-norm regularization, Acta Automatica Sinica 40 (11) (2014) 2428–2435.
[38] X. Dutoit, B. Schrauwen, J. Van Campenhout, D. Stroobandt, H. Van Brussel, M. Nuttin, Pruning and regularization in reservoir computing, Neurocomputing 72 (7–9) (2009) 1534–1546.
[39] H. Zou, T. Hastie, Regularization and variable selection via the elastic net, J. R. Stat. Soc.: Ser. B (Stat. Methodol.) 67 (2) (2005) 301–320.
[40] Z. Xu, H. Zhang, Y. Wang, X. Chang, Y. Liang, L 1/2 regularization, Sci. China Inf. Sci. 53 (6) (2010) 1159–1169.
[41] C.L.P. Chen, Z. Liu, Broad Learning System: An Effective and Efficient Incremental Learning System Without the Need for Deep Architecture, IEEE Trans. Neural Networks Learn. Syst. 29 (1) (2018) 10–24.
[42] M.M. Ebadzadeh, A. Salimi-Badr, IC-FNN: a novel fuzzy neural network with interpretable, intuitive, and correlated-contours fuzzy rules for function approximation, IEEE Trans. Fuzzy Syst. 26 (3) (2018) 1288–1302.
[43] A. Salimi-Badr, M.M. Ebadzadeh, A novel self-organizing fuzzy neural network to learn and mimic habitual sequential tasks, IEEE Trans. Cybern.
[44] P.P. Angelov, D.P. Filev, An approach to online identification of Takagi-Sugeno fuzzy models, IEEE Trans. Syst., Man, Cybern. Part B (Cybernetics) 34 (1) (2004) 484–498.
[45] H.-J. Rong, N. Sundararajan, G.-B. Huang, P. Saratchandran, Sequential adaptive fuzzy inference system (SAFIS) for nonlinear system identification and prediction, Fuzzy Sets Syst. 157 (9) (2006) 1260–1275.
[46] K. Subramanian, S. Suresh, A meta-cognitive sequential learning algorithm for neuro-fuzzy inference system, Appl. Soft Comput. 12 (11) (2012) 3603–3614.
[47] M. Pratama, S.G. Anavatti, P.P. Angelov, E. Lughofer, PANFIS: A novel incremental learning machine, IEEE Trans. Neural Networks Learn. Syst. 25 (1) (2013) 55–68.
[48] M. Pratama, S.G. Anavatti, E. Lughofer, GENEFIS: Toward an effective localist network, IEEE Trans. Fuzzy Syst. 22 (3) (2013) 547–562.
[49] M.M. Ebadzadeh, A. Salimi-Badr, CFNN: Correlated fuzzy neural network, Neurocomputing 148 (2015) 430–444.
[50] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, IEEE, 1995, pp. 1942–1948.
[51] A. Nickabadi, M.M. Ebadzadeh, R. Safabakhsh, A novel particle swarm optimization algorithm with adaptive inertia weight, Appl. Soft Comput. 11 (4) (2011) 3658–3670.
[52] A. Barua, L.S. Mudunuri, O. Kosheleva, Why trapezoidal and triangular membership functions work so well: Towards a theoretical explanation, Journal of Uncertain Systems 8.
[53] E. Hosseini-Asl, J.M. Zurada, O. Nasraoui, Deep Learning of Part-Based Representation of Data Using Sparse Autoencoders With Nonnegativity Constraints, IEEE Trans. Neural Networks Learn. Syst. 27 (12) (2016) 2486–2498.
[54] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.
[55] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. (TIST) 2 (3) (2011) 1–27.
[56] Z. Zojaji, M.M. Ebadzadeh, H. Nasiri, Semantic schema based genetic programming for symbolic regression, Appl. Soft Comput. 122 (2022) 108825.
[57] S. Zhang, B. Guo, A. Dong, J. He, Z. Xu, S.X. Chen, Cautionary tales on air-quality improvement in Beijing, Proc. R. Soc. A: Math., Phys. Eng. Sci. 473 (2205) (2017) 20170457.
[58] N. Jin, Y. Zeng, K. Yan, Z. Ji, Multivariate air quality forecasting with nested long short term memory neural network, IEEE Trans. Industr. Inf. 17 (12) (2021) 8514–8522.
[59] J.R.A. Moniz, D. Krueger, Nested LSTMs, arXiv preprint arXiv:1801.10308.
[60] K. Greff, R.K. Srivastava, J. Koutník, B.R. Steunebrink, J. Schmidhuber, LSTM: A search space odyssey, IEEE Trans. Neural Networks Learn. Syst. 28 (10) (2017) 2222–2232.
[61] H. Zheng, J. Yuan, L. Chen, Short-term load forecasting using EMD-LSTM neural networks with a Xgboost algorithm for feature importance evaluation, Energies 10 (8) (2017) 1168.
[62] H. Zang, L. Cheng, T. Ding, K.W. Cheung, Z. Liang, Z. Wei, G. Sun, Hybrid method for short-term photovoltaic power forecasting based on deep convolutional neural network, IET Generat., Transmiss. Distrib. 12 (20) (2018) 4557–4567.
[63] K. Yan, W. Li, Z. Ji, M. Qi, Y. Du, A hybrid LSTM neural network for energy consumption forecasting of individual households, IEEE Access 7 (2019) 157633–157642.
[64] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, J. Global Optimiz. 39 (3) (2007) 459–471.
[65] K. Socha, M. Dorigo, Ant colony optimization for continuous domains, Eur. J. Oper. Res. 185 (3) (2008) 1155–1173.
[66] D.T. Pham, A. Ghanbarzadeh, E. Koç, S. Otri, S. Rahim, M. Zaidi, The bees algorithm–a novel tool for complex optimisation problems, in: Intelligent production machines and systems, Elsevier, 2006, pp. 454–459.
[67] D. Simon, Biogeography-based optimization, IEEE Trans. Evolut. Comput. 12 (6) (2008) 702–713.
[68] N. Hansen, S.D. Müller, P. Koumoutsakos, Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES), Evolut. Comput. 11 (1) (2003) 1–18.
[69] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optimiz. 11 (4) (1997) 341–359.
[70] X.-S. Yang, Nature-inspired metaheuristic algorithms, Luniver Press, 2010.
[71] J.H. Holland, Genetic algorithms, Sci. Am. 267 (1) (1992) 66–73.
[72] Z.W. Geem, J.H. Kim, G.V. Loganathan, A new heuristic optimization algorithm: harmony search, Simulation 76 (2) (2001) 60–68.
[73] E. Atashpaz-Gargari, C. Lucas, Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition, in: 2007 IEEE congress on evolutionary computation, IEEE, 2007, pp. 4661–4667.
[74] A.R. Mehrabian, C. Lucas, A novel numerical optimization algorithm inspired from weed colonization, Ecol. Inform. 1 (4) (2006) 355–366.
[75] R.V. Rao, V.J. Savsani, D.P. Vakharia, Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems, Comput.-aided Des. 43 (3) (2011) 303–315.
[76] S. Bisgaard, M. Kulahci, Quality quandaries: Studying input-output relationships, part I, Q. Eng. 18 (2) (2006) 273–281.

Hamid Nasiri received the B.Sc. degree in Computer Engineering from the Semnan University, Semnan, Iran, in 2014, the M.Sc. degree in Computer Engineering from the Amirkabir University of Technology, Tehran, Iran, in 2016. He is currently a Ph.D. candidate at the Department of Computer Engineering, Amirkabir University of Technology, under the supervision of Dr. Mohammad Mehdi Ebadzadeh. His research interests include evolutionary computing, swarm intelligence, and fuzzy systems.

Mohammad Mehdi Ebadzadeh received the B.Sc. degree in Electrical Engineering from Sharif University of Technology, and his M.Sc. degree in Computer Engineering from Amirkabir University of Technology, Tehran, Iran, in 1991 and 1995 respectively. He received the PhD degree in Image and Signal Processing from TELECOM ParisTech, Paris, France, in 2004. Currently, he is a Professor in Department of Computer Engineering of Amirkabir University of Technology, Tehran, Iran. His research interests include deep reinforcement learning, computational intelligence and computational neuroscience.