A New Grouping of Rules for Time-Series Prediction by Fuzzy-induced Neural Regression

Abstract

In this paper, we present a novel grouping scheme of first-order transition rules obtained from a partitioned time-series for fuzzy-induced neural regression. The transition rules represent precedence relationships between a pair of partitions containing consecutive data points in the time-series. Here, we propose two neural network ensemble models. The first model represents a set of transition rules, each with a distinct partition in the antecedent. During the prediction phase, a certain number of pre-selected neural networks trained on the partition containing the current time-series data point are triggered. In this model, the partitions are described by their respective mid-point values during training. This induces an approximation error, as all data points lying in a partition are effectively represented by a single value. In the second model, we overcome this problem by representing each partition as a fuzzy set. This modification allows us to evaluate the degree of belongingness of a data point in each partition. Extensive experiments undertaken on the Sunspot time-series as well as on the TAIEX economic close-price time-series reveal a high prediction accuracy outperforming competitive models, thus indicating the applicability of the proposed methods to real life time-series forecasting.

Keywords

First-order transition rules, neural networks, regression, fuzzy sets, rule-based forecasting.

1. Introduction

A time-series is a discrete sequence of values obtained from a time-varying function. Generally, a time-series is one of the preferred representations for analytical study of various natural phenomena like atmospheric temperature [1], rainfall [2], seismic activity [3], population growth [4], economic growth [5] and the like. The main objective behind time-series analysis is to make accurate predictions on possible future values of the series. The task of forecasting is difficult because the actual factors affecting a time-series are manifold and often unknown. Hence, the conventional procedure adopted by researchers is to develop a model which uses values of the time-series at time instants $(t-k),\ (t-k+1),\ ...,\ t-2,\ t-1,\ t$ for some positive integer k to make a prediction on the value of the series at time instant $t+1$.

A large number of statistical methods for time-series modeling and prediction exist in the literature [6-9]. Among these models the Autoregressive Integrated Moving Average (ARIMA) [7] needs special mention. The ARIMA model combines both the autoregressive model [8] and the moving average model [9] to obtain better forecasts. It is based on three parameters, namely, *p*, the order of auto regression, *d*, the degree of differencing and *q*, the order of the moving average model. It defines the time-series value at a particular time instant as the sum of a linear combination of earlier time-series values along with a linear combination of earlier error terms (difference between predicted values and actual values). There have been many other approaches to time-series modeling developed over the years. A few well-known paradigms, which have found their way into time-series analysis, include computational intelligence models like artificial neural networks (ANN) [10] and fuzzy logic [11].

The artificial neural network (ANN) is one of the popular techniques of machine learning which has been employed by researchers for developing a variety of models for time-series prediction. In principle, the ANN is used to learn complex non-linear functions and is primarily applied to classification problems [12] and regression problems [13]. In quite a few works, the time-series has been modeled by employing a regression neural network which takes *k* time-lagged values $c(t-k)$, $c(t-k+1)$, ..., $c(t-2)$, $c(t-1)$, $c(t)$ of the time-series as input and produces the predicted value $c(t+1)$ of the time-series as output. A survey of early approaches pursued in this direction can be found in [14]. With time, more sophisticated neural models were developed to increase the accuracy of prediction on various time-series. Some of the works worth special mention are as follows. An ensemble of ARIMA and ANN was proposed in [10] to combine the advantage of the linear modeling of ARIMA and the non-linear approximations of the ANN for better forecasts. A specialized ANN architecture was developed in [15] to improve prediction accuracy on seasonal time-series, where the number of input and output neurons of the network is optimally chosen based on the duration of seasonality in the time-series. Yet another interesting work was proposed in [16], where the authors designed a recursive probabilistic extension of the well-known Levenberg-Marquardt algorithm to train recurrent neural networks for time-series modeling. Other notable works in this field include the use of recurrent neural network ensembles [17], hybrid Elman-NARX neural networks [18] and interval type-2 induced fuzzy back-propagation neural networks [19].

The advent of fuzzy set theory in the field of time-series analysis came with the works of Song and Chissom [20]-[22], where they formally provided the definition of a fuzzy time-series. A given time-series is fuzzified primarily by three steps. Firstly, the dynamic range of the time-series is divided into a number of equal or non-equal sized intervals called partitions, where each partition is a continuous section of the overall range of the series. Secondly, the dynamic

range of the series is defined as the universe of discourse and each partition obtained from the first step is represented by a fuzzy set with a corresponding fuzzy membership function defined on the universe. Finally, each data point in the time-series is assigned a linguistic value based on the partition to which it (the data point) belongs with the highest membership. Multiple types of fuzzy implication rules including first-order rules having a single antecedent and a single consequent, and higher-order rules having multiple antecedents and a single consequent can be derived from the sequence of fuzzified time-series data points. These fuzzy implication rules can then be utilized for the task of prediction. A large number of interesting works [23]-[34] on fuzzy time-series exist in the literature, many of which use advanced models incorporating first and higher order fuzzy implication relations [23], [24], heuristic functions to improve fuzzy time-series prediction [25], and multi-factor fuzzy time-series models [26].

In the present work, we propose two ensemble neural network models where a set of regression neural networks is used to predict the future value of a time-series. For each pair of consecutive time-series data points $c(t-1)$ and $c(t)$, we construct a first-order transition rule of the form $P_{t-1} \rightarrow P_t$, where P_{t-1} and P_t represent the partitions to which the data points $c(t-1)$ and $c(t)$ belong respectively. Next, in the set S of all such obtained transition rules, we identify the rules having the partition P_i in the antecedent and include them in a separate set S_i. The idea is to group the transition rules into sets where all the rules contained within a set have the same antecedent. The conditional probability $p(P_j \mid P_i)$, of partition P_j occurring as the next partition given that P_i is the current partition in the transition rule $P_i \rightarrow P_j$, is computed as the result of dividing the number of occurrences of the rule: $P_i \rightarrow P_j$ by the total number of occurrences of all rules: $P_i \rightarrow P_k, \forall k$.

In case the current time point falls in partition P_i, to predict the next possible partition, we need to fire the exhaustive set of rules containing P_i in the antecedent. This apparently is possible, if all these concurrently fire-able rules are realized on different units of hardware/software. In the present settings, we planned to realize them on neural networks. Thus for *n* concurrently fire-able rules having a common antecedent P_i, we require *n* neural networks. The question that automatically is raised: should we realize a single rule on a neural net? This, of course is un-economic. We, therefore, plan to realize a set of rules, each with a distinct partition in the antecedent, on a neural network, thereby ensuring that at a time, multiple rules on the same network cannot fire concurrently.

Any traditional supervised neural network would suffice for our present purpose. We began our study with simple feed-forward neural nets pre-trained with the back-propagation algorithm. Here the antecedent and the consequent of

the concurrently fire-able rules are used as input and output instances of the training patterns. Thus for training with *k* rules we require a set of *k* input-output training instances/patterns.

It is thus apparent that a neural network learns a subset of the set of all rules where each rule in the subset contains distinct partitions in the antecedent. However, it may not realize the exhaustive set of rules with all possible partitions in the antecedent. In other words, consider a neural net realizing three rules with partitions P_2, P_6 and P_7 in the antecedents of these three rules. The entire set of rules may however include all the partitions P_1 through P_{10} in the antecedent. Naturally, when the current partition is P_3, we would not use this neural net as P_3 does not appear in the antecedent of any of the rules realized by the neural network. This calls for a pre-selector that selects the right neural networks depending on the occurrence of the partition corresponding to the current data point in the time-series as antecedent of one rule realized by the networks. We here used a Radial Basis Function (RBF) [35] neural network to serve the purpose. The RBF network contains a hidden layer neuron corresponding to each distinct antecedent on which the neural network concerned is pre-trained. The neural network is triggered only when at least one of the hidden neurons in the pre-selector RBF is fired indicating that the current input partition occurs as the antecedent in a transition rule realized by the neural net. This ensures that only those networks which have been trained on rules containing the current partition as antecedent are triggered for prediction.

Although a time-series partition describes a bounded interval of a parameter (like temperature), it is generally represented by an approximate single value, such as the center of the interval. Such representation introduces an approximation error for all data points lying in the interval. One approach to overcome the creeping of approximation error is to describe each partition by a fuzzy set that takes into account, the distance between each data point and its respective partition-center while assigning memberships to the data points. These membership values indicate the degree of belongingness of the data points in a given partition. In the previous approach, we train each neural network in the ensemble with transition rules where the antecedents and consequents represent mid-point or center values of partitions. As the fuzzy representation for a partition, explained above, is relatively better than its mid-point representation, we attempt to incorporate the fuzzy representation in our neural network ensemble model. This can be done by fuzzifying the first-order transition rules where the antecedent of each rule is replaced by a membership function. The input instances in this case, for a neural net in the ensemble are the fuzzy membership values of the current time-series data point in every partition. Further, considering the effect of each partition in the antecedent of a transition rule eliminates the requirement of a pre-selector logic.

Experiments have been carried out to check the efficacy of the proposed models on real life chaotic time-series like the Sunspot [36] for comparison with existing neural network based time-series models. Also, the proposed models have also been applied on the Taiwan Stock Exchange Index (TAIEX) [37] economic close price time-series for the period 1990-2004 for comparison with existing fuzzy time-series models. The experiments indicate that the proposed models outperform the best among the existing models by a relatively large margin.

The remainder of this paper is organized as follows. Section 2 provides a basic overview on fuzzy sets, the back-propagation training algorithm for neural networks and radial basis function networks. Section 3 discusses the first proposed model based on discrete first-order transition rules. Section 4 deals with the second proposed model applying fuzzy partitioning schemes for training. Section 5 contains the experiments carried out on various time-series as well as comparisons with other existing models. Conclusions are listed in Section 6.

2. Preliminaries

In this section, we discuss on some of the well-known concepts required for understanding the rest of the paper. Specifically, the definitions of fuzzy sets, membership functions and time-series partitioning are provided along with a brief description of the back-propagation training algorithm for the neural network as well as Radial Basis Function networks.

2A. Fuzzy sets and time-series partitioning

A fuzzy set is a generalization of a conventional set where each element belongs with a certain membership value lying in the range [0,1] unlike a conventional set where each element belongs with a membership value which is either 0 (does not belong to the set) or 1 (belongs to the set). A fuzzy set is formally defined in Definition 1.

Definition 1: Given a set *U* as the "universe of discourse", a **fuzzy set** *A* on the universe *U* is defined as a set of 2-tuples as shown below

$$A = \{(x, \mu_A(x)) \mid x \in U\} \tag{1}$$

where *x* is an element of the universe *U* and $\mu_A(x) \in [0,1]$ denotes the membership value of the element *x* in the fuzzy set *A*. If the universe of discourse *U* is continuous and infinite, it is not possible to define a set of discrete 2-tuples as given in equation (1) for the fuzzy set *A*. In such cases, a **fuzzy membership function** $\mu_A : U \rightarrow [0,1]$ is defined to map each element in the universe of discourse to its corresponding membership value in the fuzzy set *A*.

Definition 2: A **time-series** is a discrete sequence of values sampled from a temporally varying measurable entity like atmospheric temperature, rainfall, population growth and the like. A time-series can be considered as a vector of length *n* as shown below:

$$\vec{C} = [c(t-(n-1)k),\ c(t-(n-2)k),\ ...,\ c(t-2k),\ c(t-k),\ c(t)] = [c_1, c_2, c_3, ..., c_n] \tag{2}$$

where $c_i = c(t-(n-i)k)$ represents the i^{th} sample of the series and *k* is the sampling interval.

Definition 3: Given a time-series $\vec{C}$, let c_{max} and c_{min} represent the maximum and minimum values of the time-series respectively. We define **partitioning** as the act of dividing the range $R = [c_{min}, c_{max}]$, into *h* non-overlapping contiguous intervals $P_1, P_2, ..., P_h$ such that the following two conditions jointly hold:

$$P_i \cap P_j = \varnothing,\ \forall i \neq j,\ i, j \in \{1, 2, ..., h\} \tag{3}$$

$$\bigcup_{i=1}^{h} P_i = R,\ \text{for integer } i. \tag{4}$$

Let a partition $P_i = [P_i^-, P_i^+]$ be defined as a bounded set where P_i^- and P_i^+ represent the lower and upper bounds of the partition respectively. Clearly, a time-series data point $c(t)$ belongs to the partition P_i if and only if $P_i^- \leq c(t) \leq P_i^+$.

Definition 4: Let $c(t)$ and $c(t+1)$ denote two consecutive data points in a time-series $\vec{C}$ and let P_i and P_j be the partitions to which these data points belong. Then we denote $P_i \rightarrow P_j$ as a **first-order transition rule** that exists in the time-series $\vec{C}$, with the antecedent P_i and consequent P_j. The following example demonstrates the first-order transition rule in a time-series.

Example 1: Let a given time-series $\vec{C} = [3,5,10,8,3,1,9,2]$, be divided into five partitions $P_1 = [0,2),\ P_2 = [2,4),\ P_3 = [4,6),\ P_4 = [6,8),\ P_5 = [8,10]$. Clearly, the sequence of partitions corresponding to each data point in the time-series is $\vec{P} = [P_2, P_3, P_5, P_5, P_2, P_1, P_5, P_2]$. From the sequence $\vec{P}$, the first-order transition rules obtained, are $P_2 \rightarrow P_3,\ P_3 \rightarrow P_5,\ P_5 \rightarrow P_5,\ P_5 \rightarrow P_2,\ P_2 \rightarrow P_1,\ P_1 \rightarrow P_5,\ P_5 \rightarrow P_2$.

2.B. Back-propagation Algorithm

One of the most well-known training algorithms employed to optimize the weights of a neural network is the back-propagation algorithm. The basic principle behind the back-propagation algorithm is to minimize a loss function by iterative modification of the weights of the network over all the training examples. The algorithm uses gradient descent learning on an error (energy) function of weights, where the error function is designed to minimize the Euclidean norm of two vectors: targeted output vector and computed output vector. Here the vectors are defined with respect to the output signals corresponding to the neurons in the last layer. The algorithm begins by randomly initializing the weights of a neural network and has two primary steps which are iteratively carried out over all the training examples until the network performance is satisfactory:

Step 1. Forward Propagation: In this step, a training example is input in order to obtain the output activation values from the network and compute the loss or error metric e of the network output with respect to the desired output.

Step 2. Backward Propagation and Weight Update: This step is used to compute the gradient $\frac{\partial e}{\partial w}$ of the error (or loss) with respect to each and every weight *w* in the network. Furthermore, for a weight *w*, the negative of the gradient $\frac{\partial e}{\partial w}$ represents the direction of decreasing loss. Hence, the weight *w* is updated as follows:

$$w_i \leftarrow w_{i-1} - \alpha\left(\frac{\partial e_{i-1}}{\partial w_{i-1}}\right) \tag{5}$$

where w_i and w_{i-1} represent the weight at the i^{th} and $(i\text{-}1)^{th}$ iterations respectively, e_{i-1} is the error at the $(i\text{-}1)^{th}$ iteration, $\frac{\partial e_{i-1}}{\partial w_{i-1}}$ is the value of the gradient and α denotes the learning rate of the network.

2.C. Radial Basis Function (RBF) Networks

A Radial Basis Function (RBF) network is a 3-layer (input, hidden and output) neural network which uses non-linear radial basis functions as activation functions in its hidden layer. The output of the network is generally a linear combination of the hidden layer activation values. Let the input to the network be represented as a vector $\vec{X} = (x_1, x_2, ..., x_k)$ and let there be *h* hidden layer neurons in the network. The output of the RBF network is then given as:

$$\phi(\vec{X}) = \sum_{i=1}^{h} a_i \rho\left(\left\|\vec{X} - \vec{C}_i\right\|\right) \tag{6}$$

where $\vec{C}_i$ represents the central vector of the i^{th} hidden layer neuron and the norm $\left\|\vec{X}-\vec{C}_i\right\|$, is a distance measure like the Euclidean distance between the vectors $\vec{X}$ and $\vec{C}_i$. Intuitively, each hidden layer neuron outputs the similarity between the input vector and its own central vector. Hence, a commonly used radial basis function is the Gaussian function as given below:

$$\rho(\left\|\vec{X}-\vec{C}_i\right\|)=e^{-\beta(\left\|\vec{X}-\vec{C}_i\right\|)^2} \tag{7}$$

where β is a positive constant. Due to the dependence of the activation function on the distance between the input vector and a hidden neuron's central vector, the function is radially symmetric about the central vector of the neuron. In this paper, we use RBF neurons as pre-selectors to determine the neural networks to be triggered based on the transition rules on which the networks have been pre-trained.

3. First-order transition rule based NN model

In this section, we propose a neural network ensemble model trained on first-order transition rules obtained from a given time-series. Let *S* be the set of all first-order transition rules and let S_i denote the set of rules having the partition P_i in the antecedent. Further, let $p(P_j/P_i)$ indicate the conditional probability of occurrence of P_j as the next partition, given that P_i is the current partition. Clearly, the value of $p(P_j/P_i)$ is computed as given below:

$$p(P_j/P_i)=\frac{count(P_i\rightarrow P_j)}{\sum_{\forall k} count(P_i\rightarrow P_k)} \tag{8}$$

where $count(P_i\rightarrow P_j)$ represents the number of occurrences of the transition rule $P_i\rightarrow P_j$. The set S_i can then be represented as follows:

$$S_i=\{P_i\rightarrow P_k \mid p(P_k/P_i)>0\}. \tag{9}$$

Given the current partition P_i, in order to make a prediction for the next partition, it is desirable to consider the weighted contribution of all the transition rules having P_i in the antecedent. This can be best achieved by designing a model that allows for the simultaneous firing of all the rules in the set S_i. In order to realize such a model, we use an ensemble of feed-forward neural networks which satisfies the following conditions:

Condition 1: Given the current input partition P_i, each neural network in the ensemble can fire at most a single transition rule having P_i in the antecedent.

Condition 2: All the transition rules having partition P_i in the antecedent must be fired by the ensemble. In other words, all the rules contained in the set S_i must be fired.

Condition 3: No two neural networks in the ensemble can fire the same transition rule, i.e., for any two rules $P_i \rightarrow P_a$ and $P_i \rightarrow P_b$ fired simultaneously, $P_a \neq P_b$.

The above mentioned conditions can be satisfied by imposing certain constraints on the training sets of the neural networks. A few observations that are evident from the above conditions are given as follows:

Theorem 1: For any neural network NN_i in the ensemble, its training set T_i cannot contain multiple transition rules having the same partition in the antecedent.

Proof. We prove this theorem by the method of contradiction. Let the training set T_i contain two transition rules $P_a \rightarrow P_b$ and $P_a \rightarrow P_c$ having the same partition P_a in the antecedent. If the current input partition is P_a, following Condition 1, any one of the rules $P_a \rightarrow P_b$ or $P_a \rightarrow P_c$ will be fired by the network NN_i. Without any loss of generality, let us arbitrarily assume that the fired rule is $P_a \rightarrow P_b$. Hence, the rule $P_a \rightarrow P_c$ is not fired. By Condition 2, we can say that the rule $P_a \rightarrow P_c$ is fired by some other neural network NN_j in the ensemble. However, the decision to fire $P_a \rightarrow P_b$ is completely arbitrary and the rule $P_a \rightarrow P_c$ could have been fired as well, violating Condition 3. Hence, we arrive at a contradiction and our initial assumption was incorrect. The training set T_i does not contain multiple rules having the same partition in the antecedent. □

Theorem 2: For any two neural networks NN_i and NN_j in the ensemble having training sets T_i and T_j respectively, the training sets are disjoint, i.e., $T_i \cap T_j = \varnothing$.

Proof. We prove this theorem by the method of contradiction. Let us assume that the training sets T_i and T_j of the two neural networks NN_i and NN_j contain the same transition rule $P_a \rightarrow P_b$. Hence, when the current input partition is P_a, both the networks NN_i and NN_j fire the same rule $P_a \rightarrow P_b$ as, from Theorem 1, they have no other rule in their training sets with P_a in the antecedent. This however, violates Condition 3 mentioned above thereby contradicting our

initial assumption that the two training sets have a transition rule which is common. Thus, all the training sets are completely disjoint with respect to one another. □

With the above conditions in mind, we group the set *S* of transition rules into training sets using the following steps:

Step 1. The set *S* is divided into groups or subsets of transition rules where all the rules in a group have the same partition in the antecedent. Let the subset of rules having P_i in the antecedent be denoted by S_i following equation (9). Clearly, the number of such subsets is equal to the total number of distinct partitions occurring in the antecedent of the transition rules.

Step 2. The transition rules in each subset are ordered (following any arbitrary order). Now, the training set for the first neural net in the ensemble is constructed by collecting the transition rules which occur in the first position in each subset, the training set for the second neural net is constructed by collecting the transition rules occurring in the second position in each subset and so on.

The algorithm for the construction of training sets is formally presented in Pseudo Code 1. Also, for better comprehension, an illustrative example of the same is provided in Example 2.

Pseudo Code 1: Training set construction algorithm

Input: A set *S* of all first-order transition rules obtained from a time-series.

Output: A sequence of training sets $T_1, T_2, ..., T_v$ where the i^{th} training set T_i contains the transition rules on which the i^{th} neural network NN_i is trained.

```
BEGIN
  Let k be the number of distinct partitions occurring in the antecedent of the transition rules in S;
  Group the transition-rules in S into k arrays S_1, S_2, ..., S_k such that the array S_i, i ∈ {1,2,...,k} contains only
  the rules having partition P_i in the antecedent;
  j ← 1; //denotes the training set number
  WHILE construction of training sets is not complete, DO
  BEGIN
    Initialize T_j ← ∅; // T_j is initially an empty set
    FOR each array S_i, i ∈ {1,2,...,k} DO
    BEGIN
      Let P_x → P_y be the first rule (if any) in the array S_i;
      T_j ← T_j ∪ {P_x → P_y}; //The rule is added to the training set
      Remove the rule P_x → P_y from the array S_i;
    END FOR;
    IF all the arrays S_1, S_2, ..., S_k are empty, THEN
      Break from WHILE loop; //Training set construction is complete
    END IF;
    j ← j+1; // Training set T_j has been constructed, moving to construction of T_{j+1}
  END WHILE;
END
```

Example 2: Let the transition rules obtained from the time-series be represented as the set *S* as given below:

$$S = \{P_1 \rightarrow P_1,\ P_1 \rightarrow P_2,\ P_2 \rightarrow P_1,\ P_2 \rightarrow P_2,\ P_2 \rightarrow P_3,\ P_3 \rightarrow P_1,\ P_3 \rightarrow P_3\}. \quad (10)$$

First, we group the rules in *S* according to the common antecedents. Hence, we obtain three ordered subsets of *S*, i.e., $S_1 = \{P_1 \rightarrow P_1,\ P_1 \rightarrow P_2\}$, $S_2 = \{P_2 \rightarrow P_1,\ P_2 \rightarrow P_2, P_2 \rightarrow P_3\}$ and $S_3 = \{P_3 \rightarrow P_1,\ P_3 \rightarrow P_3\}$. For the construction of the first training set T_1, we collect the first transition rules in each set S_1, S_2 and S_3. Hence, the training set T_1 is constructed as follows:

$$T_1 = \{P_1 \rightarrow P_1,\ P_2 \rightarrow P_1,\ P_3 \rightarrow P_1\}. \quad (11)$$

The training sets T_2 and T_3 contain the second and third transition rules in S_1, S_2 and S_3 and are given as follows:

$$T_2 = \{P_1 \rightarrow P_2,\ P_2 \rightarrow P_2,\ P_3 \rightarrow P_3\} \quad (12)$$

$$T_3 = \{P_2 \rightarrow P_3\}. \quad (13)$$

It should be noted that the training set T_3 contains only a single transition rule as there are no rules left in the sets S_1 and S_3 after the construction of the training sets T_1 and T_2. □

Having constructed the training sets, we are ready to train the neural nets in the ensemble using the backpropagation algorithm. It should be noted that the training sets contain transition rules where both the antecedent and consequent of each rule represents a partition label like P_1. However, our current task demands a computational model whose expected input is the current time-series data point and the desired output is the next or future time-series data point, both of which are real numbers. This requires us to use regression neural networks in the ensemble whose input and output values are both real numbers. Hence, in order to appropriately convert the training set to fit our needs, we replace each partition label P_i with its corresponding partition mid-value m_i. With this modification to the training sets in place, we are able to train the neural networks in the ensemble.

During the prediction phase, let the partition containing the current time-series data point (input to the system) be P_i. It may so happen, that a neural network in the ensemble has not been trained on a rule having P_i in the antecedent. Clearly, it is undesirable to use such a network for prediction as that may lead to erroneous results. In order to deal with this problem, we use a set of RBF neurons as pre-selectors for triggering appropriate neural networks in the ensemble based on the given input. Let the total number of partitions be *h*. We then construct *h* RBF neurons, one for each partition. The output activation value ρ_i for the i^{th} RBF neuron is given as follows:

$$\rho_i = e^{-\beta(|x^2 - m_i^2|)} \tag{14}$$

where x is the input to the RBF neuron, m_i is the mid-point value of the i^{th} partition P_i and β is an empirically chosen constant. Intuitively, the i^{th} RBF neuron outputs the similarity between the input and the mid-value of the i^{th} partition P_i. Given the current input partition P_c, our objective is to trigger only those neural nets which have been trained on at least one rule having P_c in the antecedent. In other words, a neural network NN_j is enabled, if the current input partition P_c is present in the antecedent of any one of the training set rules for NN_j. Hence, the enable signal for NN_j is obtained by a logical OR operation of the outputs of the RBF neurons which correspond to the partitions occurring in the antecedent of the training set rules for NN_j. The operands for logical OR being binary, we have to apply a simple thresholding technique to the outputs of the RBF neurons to convert them to binary values. An illustrative example of the pre-selection technique is given in Example 3.

Example 3: Let us consider the transition rules given in equation (10) of Example 2. There are three partitions in the rules, i.e., P_1, P_2 and P_3. We construct three RBF neurons RBF_1, RBF_2 and RBF_3 corresponding to the partitions P_1, P_2 and P_3 respectively. Now, the training set $T_1 = \{P_1 \rightarrow P_1,\ P_2 \rightarrow P_1,\ P_3 \rightarrow P_1\}$ for the neural net NN_1 contains all the three partitions P_1, P_2 and P_3 in the antecedents of its transition rules. Hence, the enable signal for NN_1 is obtained by a logical OR of the outputs of RBF_1, RBF_2 and RBF_3. Similarly, the neural net NN_2 is also enabled by a logical OR of the outputs of RBF_1, RBF_2 and RBF_3. However, the training set $T_3 = \{P_2 \rightarrow P_3\}$ for the neural net NN_3 contains only the partition P_2 in the antecedent. Hence, the enable signal for NN_3 can be directly obtained from the output of RBF_2. □

The final prediction $c'(t+1)$ for the next time-series data point, given the current data point $c(t)$, is computed as the weighted sum of the individual outputs obtained from the neural nets which are triggered by the pre-selector RBF neurons. Let there be v selected neural nets and let their respective outputs be $o_1, o_2, ..., o_v$ lying in the partitions $P_{o1}, P_{o2}, ..., P_{ov}$ respectively. The final output of the ensemble is then given as follows:

$$c'(t+1) = \sum_{i=1}^{v} \left(o_i \times p\left(P_{oi} \middle/ P_c \right) \right) \tag{15}$$

where P_c is the partition containing the current time-series data point $c(t)$ and $p\left(P_{oi}/P_c\right)$ is the conditional probability of occurrence of P_{oi} as the next partition, given that the current partition is P_c. A schematic block diagram of the steps in our proposed approach is illustrated in Figure 1. The architecture of the ensemble corresponding to the set of transition rules given in Example 1 is shown in Figure 2. It should be noted that in our paper, we have designed each neural network in the ensemble with a single hidden layer having 10 hidden neurons as shown in Figure 3.

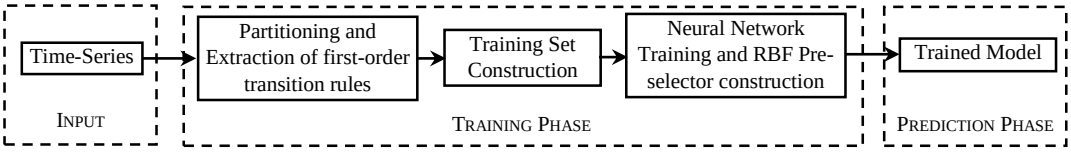

Fig. 1. Schematic block diagram of the steps proposed in the neural network ensemble model

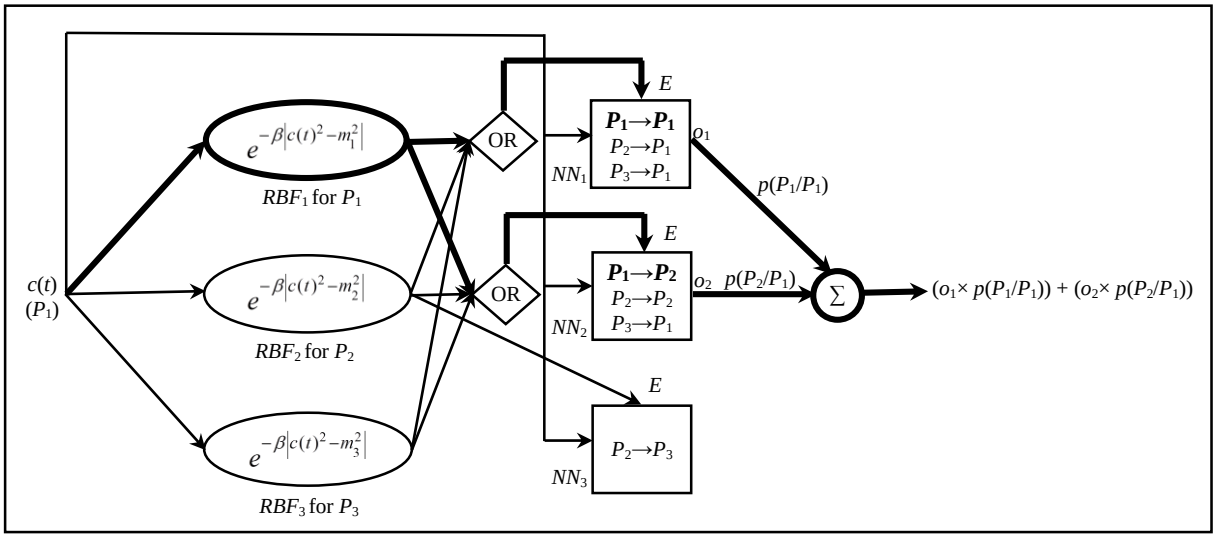

Fig. 2. Architecture of the ensemble corresponding to Example 1, where m_1, m_2 and m_3 are the mid-values of the partitions P_1, P_2 and P_3 respectively, the given input lies in partition P_1, hence, RBF_1 is fired and the corresponding rules in NN_1 and NN_2 are fired. The fired paths and rules are shown in bold. The enable signals are marked with the symbol *E*.

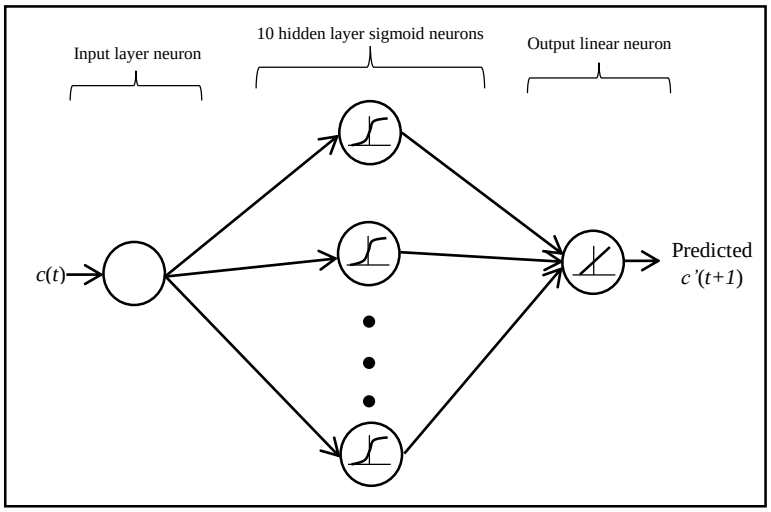

Fig. 3.Neural network architecture for the first-order transition rule based NN model, $c(t)$ is the input time-series value and $c'(t+1)$ is the predicted future value by the neural network.

4. Fuzzy Rule Based NN Model

In this section, we propose a variation in the training scheme of the neural nets from the previous model. It should be noted that a time-series data point cannot be wholly represented by a partition. A partition is a certain continuous portion of the dynamic range of the time-series. The conventional method to represent a partition is by its mid-point. Hence, by approximating a data point lying in a partition with its corresponding mid-point value, we intentionally delve into some approximation error which grows with the width of the partition itself. A way to avoid this error is to consider each partition as a fuzzy set, with the dynamic range of the time-series representing the universe of discourse. Clearly, each partition is then associated with a judiciously chosen membership function and every time-series data point will have some membership value lying in the set [0,1], for a partition.

We want to design a membership function with the peak corresponding to the center of the partition and a gradual fall-off on either side of the peak with a value almost close to 0 towards the bounds of the partition. The membership value need not however necessarily be 0 at the boundaries of the partition. One typical function which satisfies this requirement is the Gaussian membership function as shown in Figure 4. We use these membership values to train the neural networks in this model. The advantage of this approach is the exploitation of the inherent fuzziness involved in the task of assigning a partition to a time-series data point and using it for better prediction results.

Let the i^{th} partition be $p_i = [l_i, u_i]$, and let the mid-point of the partition be m_i. We define the standard deviation for the Gaussian membership function as $\sigma_i = (u_i - m_i) = (m_i - l_i)$. With the above terms defined, the membership function corresponding to the i^{th} partition is given in equation (16) and illustrated in Figure 4.

$$\mu_i(x) = e^{-\frac{(x-m_i)^2}{2\sigma_i^2}} \tag{16}$$

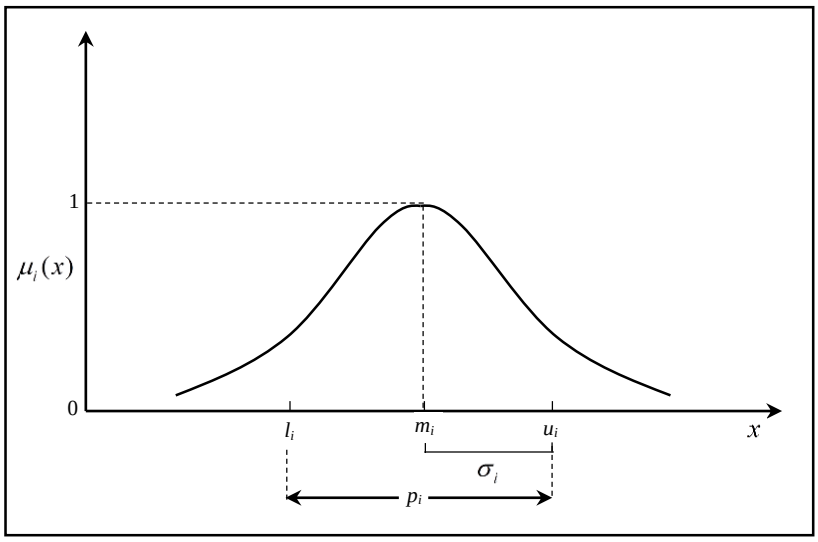

Fig. 4. Membership function $\mu_i(x)$ corresponding to the partition p_i. The symbols l_i, and u_i represent the lower and upper bounds of the partition p_i and m_i is the corresponding mid-point of the partition.

Let there be k partitions in the time-series. For each first-order transition rule of the form $p_i \rightarrow p_j$, (p_i and p_j represent partitions) a training rule of the form $(\mu_1(m_i), \mu_2(m_i), ..., \mu_k(m_i)) \rightarrow m_j$ is created, where m_i and m_j are the mid-points of partitions p_i and p_j respectively and $\mu_l(m_i),\ l \in \{1, 2, ..., k\}$, are the membership values of m_i in each partition of the time-series. These modified rules are used to train the neural network after they have been grouped following the grouping algorithm mentioned in Pseudo Code 1.

It is worth noting here that in each of the modified training rules, membership values of all the partitions are present in the antecedent. This relieves the model from the necessity of a pre-selector logic. A schematic of the proposed model is shown in Figure 5 and the architecture of the respective neural networks is given in Figure 6.

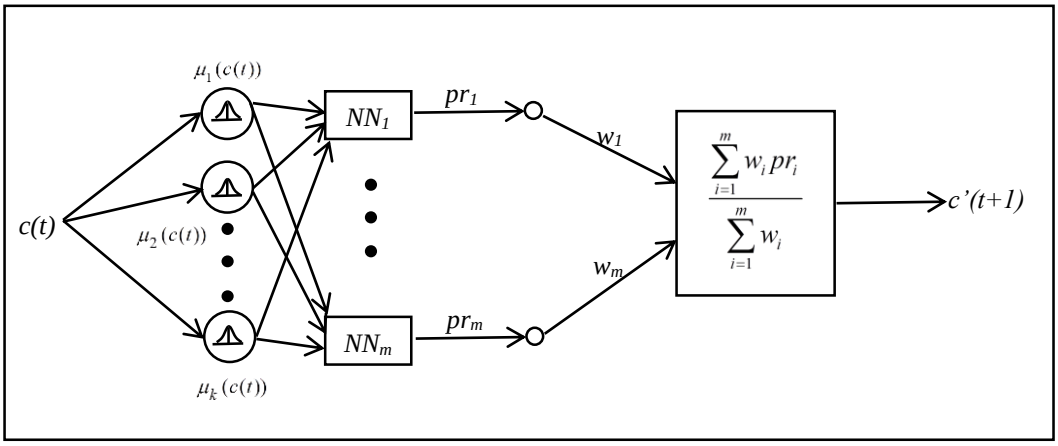

Fig. 5. Schematic diagram of the fuzzy rule based neural network model (considering k partitions), $c(t)$ is the input time-series data point, $\mu_i(c(t))$ denotes the membership value of $c(t)$ in the i[th] partition, NN_i is the i^{th} neural network producing the prediction pr_i attached with weight w_i and $c'(t+1)$ represents the predicted time-series value.

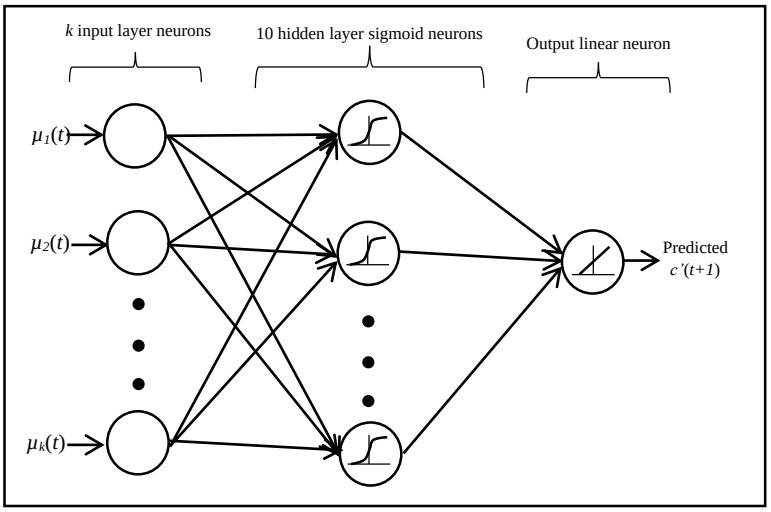

Fig. 6. Neural network architecture for the fuzzy transition rule based model, input layer neurons correspond to the membership value of a time-series data point $c(t)$ in each of the k partitions.

5. Experiments and Results

In this section, we present the results obtained from two experiments carried out to test the prediction efficacy of the proposed models. In the first experiment, we apply our models to the well-known Sunspot [36] time-series data from November 1834 to June 2001 for comparison with existing neural time-series prediction models in the literature. In the second experiment, we use the TAIEX [37] economic close-price time-series for prediction in the field of trading and investment as well as for comparison with existing fuzzy time-series models. Both the experiments are performed

using MATLAB on an Intel Core i7 processor with a clock speed of 2.3GHz and 8 GB of RAM. The details of each experiment are provided in the following sub-sections.

5.1 Experiment 1: Sunspot time-series prediction

The sunspot time-series data is used to record solar activity and is a classic example of a real-life chaotic time-series. Due to the effect of solar activity on various aspects of human life like the climate and weather, prediction of the sunspot time-series has become an important and critical challenge for many researchers. In this paper, we use the smoothed sunspot time-series data from the World Data Center for sunspot index. The time-series from the period of November 1834 to June 2001 has 2000 data-points which is divided into two equal parts of 1000 points each. The first half is used for training the models and the second half is used for testing. The time-series is first scaled to the range [0,1] and the following steps are carried out for the experiment:

Step 1. Partitioning, First-order rule extraction and neural network training: In this step, we first partition the training phase time-series into 20 partitions. We experimentally choose the value 20 based on best prediction results. The first-order transition rules extracted from the partitioned time-series along with the probability of occurrence of each rule is given in the transition matrix shown in Table 1. It should be noted that the entry corresponding to the cell (P_i, P_j) contains the probability of occurrence of the first-order transition $P_i \rightarrow P_j$.

Table 1
Transition matrix obtained for the first-order transition rules from Sunspot training phase time-series

To Partition / From partition	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	P_{11}	P_{12}	P_{13}	P_{14}	P_{15}	P_{16}	P_{17}	P_{18}	P_{19}	P_{20}
P_1	0.7561	0.2098	0.0293	0.0049	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P_2	0.3909	0.3182	0.2	0.0727	0.0091	0.0091	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P_3	0.0522	0.2708	0.3333	0.1979	0.0729	0.0625	0	0.0104	0	0	0	0	0	0	0	0	0	0	0	0
P_4	0.0233	0.0465	0.1860	0.2558	0.2209	0.1163	0.1047	0.0233	0.0116	0.0116	0	0	0	0	0	0	0	0	0	0
P_5	0	0.0098	0.0882	0.2255	0.2549	0.2255	0.1471	0.0392	0	0	0	0	0	0.0098	0	0	0	0	0	0
P_6	0	0.0108	0.0645	0.0860	0.2581	0.2366	0.1398	0.1398	0.0215	0.0430	0	0	0	0	0	0	0	0	0	0
P_7	0	0	0.0270	0.0405	0.1892	0.2162	0.2568	0.1351	0.1081	0.0135	0.0135	0	0	0	0	0	0	0	0	0
P_8	0	0	0.0164	0.0328	0.1475	0.1148	0.1639	0.1803	0.1148	0.0984	0.0492	0.0492	0.0164	0.0164	0	0	0	0	0	0
P_9	0	0	0.0233	0	0.0233	0.0233	0.1163	0.1860	0.2093	0.2326	0.0698	0.0930	0.0233	0	0	0	0	0	0	0
P_{10}	0	0	0	0	0.0244	0.0976	0.0244	0.1463	0.1463	0.2927	0.1707	0.0244	0.0488	0.0244	0	0	0	0	0	0
P_{11}	0	0	0	0	0	0.1034	0.0345	0.1379	0.1034	0.1724	0.2069	0.1034	0	0	0.1379	0	0	0	0	0
P_{12}	0	0	0	0	0	0	0.1250	0.0625	0.1250	0.0625	0.0625	0.1250	0.25	0.0625	0.0625	0	0	0	0	0.0625
P_{13}	0	0	0	0	0	0	0	0	0.2222	0.0556	0.2778	0.0556	0.2222	0.1111	0.0555	0	0	0	0	0
P_{14}	0	0	0	0	0	0	0	0.1111	0.1111	0	0.2222	0.1111	0.1111	0.1111	0.1111	0.1111	0	0	0	0
P_{15}	0	0	0	0	0	0	0	0	0	0	0.1250	0.1250	0.1250	0	0.1250	0.1250	0.3750	0	0	0
P_{16}	0	0	0	0	0	0	0	0	0	0	0	0	0.5	0.5	0	0	0	0	0	0
P_{17}	0	0	0	0	0	0	0	0	0	0	0	0	0.75	0.25	0	0	0	0	0	0
P_{18}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
P_{19}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P_{20}	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0

The extracted rules are then grouped into training sets, each representing a mapping of distinct antecedents to consequents following the algorithm given in pseudo code 1. The groups of transition rules used to train each neural network is shown in Table 2. It should be noted that groups with less than 6 transition rules are ignored for training purposes.

Table 2
Groups of first-order transition rules used to train each neural network (NN_1 to NN_9) for the sunspot time-series. NN_{10} to NN_{12} are not trained as the number of rules obtained for their respective training sets is below the minimum threshold of 6.

Antecedent	NN_1	NN_2	NN_3	NN_4	NN_5	NN_6	NN_7	NN_8	NN_9	NN_{10}	NN_{11}	NN_{12}
P_1	$P_1 \rightarrow P_1$	$P_1 \rightarrow P_2$	$P_1 \rightarrow P_3$	$P_1 \rightarrow P_4$								
P_2	$P_2 \rightarrow P_1$	$P_2 \rightarrow P_2$	$P_2 \rightarrow P_3$	$P_2 \rightarrow P_4$	$P_2 \rightarrow P_5$	$P_2 \rightarrow P_6$						
P_3	$P_3 \rightarrow P_1$	$P_3 \rightarrow P_2$	$P_3 \rightarrow P_3$	$P_3 \rightarrow P_4$	$P_3 \rightarrow P_5$	$P_3 \rightarrow P_6$	$P_3 \rightarrow P_8$					
P_4	$P_4 \rightarrow P_1$	$P_4 \rightarrow P_2$	$P_4 \rightarrow P_3$	$P_4 \rightarrow P_4$	$P_4 \rightarrow P_5$	$P_4 \rightarrow P_6$	$P_4 \rightarrow P_7$	$P_4 \rightarrow P_8$	$P_4 \rightarrow P_9$	$P_4 \rightarrow P_{10}$		
P_5	$P_5 \rightarrow P_2$	$P_5 \rightarrow P_3$	$P_5 \rightarrow P_4$	$P_5 \rightarrow P_5$	$P_5 \rightarrow P_6$	$P_5 \rightarrow P_7$	$P_5 \rightarrow P_8$	$P_5 \rightarrow P_{14}$				
P_6	$P_6 \rightarrow P_2$	$P_6 \rightarrow P_3$	$P_6 \rightarrow P_4$	$P_6 \rightarrow P_5$	$P_6 \rightarrow P_6$	$P_6 \rightarrow P_7$	$P_6 \rightarrow P_8$	$P_6 \rightarrow P_9$	$P_6 \rightarrow P_{10}$			
P_7	$P_7 \rightarrow P_3$	$P_7 \rightarrow P_4$	$P_7 \rightarrow P_5$	$P_7 \rightarrow P_6$	$P_7 \rightarrow P_7$	$P_7 \rightarrow P_8$	$P_7 \rightarrow P_9$	$P_7 \rightarrow P_{10}$	$P_7 \rightarrow P_{11}$			
P_8	$P_8 \rightarrow P_3$	$P_8 \rightarrow P_4$	$P_8 \rightarrow P_5$	$P_8 \rightarrow P_6$	$P_8 \rightarrow P_7$	$P_8 \rightarrow P_8$	$P_8 \rightarrow P_9$	$P_8 \rightarrow P_{10}$	$P_8 \rightarrow P_{11}$	$P_8 \rightarrow P_{12}$	$P_8 \rightarrow P_{13}$	$P_8 \rightarrow P_{14}$
P_9	$P_9 \rightarrow P_3$	$P_9 \rightarrow P_5$	$P_9 \rightarrow P_6$	$P_9 \rightarrow P_7$	$P_9 \rightarrow P_8$	$P_9 \rightarrow P_9$	$P_9 \rightarrow P_{10}$	$P_9 \rightarrow P_{11}$	$P_9 \rightarrow P_{12}$	$P_9 \rightarrow P_{13}$		
P_{10}	$P_{10} \rightarrow P_5$	$P_{10} \rightarrow P_6$	$P_{10} \rightarrow P_7$	$P_{10} \rightarrow P_8$	$P_{10} \rightarrow P_9$	$P_{10} \rightarrow P_{10}$	$P_{10} \rightarrow P_{11}$	$P_{10} \rightarrow P_{12}$	$P_{10} \rightarrow P_{13}$	$P_{10} \rightarrow P_{14}$		
P_{11}	$P_{11} \rightarrow P_6$	$P_{11} \rightarrow P_7$	$P_{11} \rightarrow P_8$	$P_{11} \rightarrow P_9$	$P_{11} \rightarrow P_{10}$	$P_{11} \rightarrow P_{11}$	$P_{11} \rightarrow P_{12}$	$P_{11} \rightarrow P_{15}$				
P_{12}	$P_{12} \rightarrow P_7$	$P_{12} \rightarrow P_8$	$P_{12} \rightarrow P_9$	$P_{12} \rightarrow P_{10}$	$P_{12} \rightarrow P_{11}$	$P_{12} \rightarrow P_{12}$	$P_{12} \rightarrow P_{13}$	$P_{12} \rightarrow P_{14}$	$P_{12} \rightarrow P_{15}$	$P_{12} \rightarrow P_{20}$		
P_{13}	$P_{13} \rightarrow P_9$	$P_{13} \rightarrow P_{10}$	$P_{13} \rightarrow P_{11}$	$P_{13} \rightarrow P_{12}$	$P_{13} \rightarrow P_{13}$	$P_{13} \rightarrow P_{14}$	$P_{13} \rightarrow P_{15}$					
P_{14}	$P_{14} \rightarrow P_8$	$P_{14} \rightarrow P_9$	$P_{14} \rightarrow P_{11}$	$P_{14} \rightarrow P_{12}$	$P_{14} \rightarrow P_{13}$	$P_{14} \rightarrow P_{14}$	$P_{14} \rightarrow P_{15}$	$P_{14} \rightarrow P_{16}$				
P_{15}	$P_{15} \rightarrow P_{11}$	$P_{15} \rightarrow P_{12}$	$P_{15} \rightarrow P_{13}$	$P_{15} \rightarrow P_{15}$	$P_{15} \rightarrow P_{16}$	$P_{15} \rightarrow P_{17}$						
P_{16}	$P_{16} \rightarrow P_{13}$	$P_{16} \rightarrow P_{14}$										
P_{17}	$P_{17} \rightarrow P_{13}$	$P_{17} \rightarrow P_{14}$										
P_{18}	$P_{18} \rightarrow P_{17}$											
P_{19}												
P_{20}	$P_{20} \rightarrow P_{18}$											

The bunched first-order transition rules are further processed to yield training sets for the two proposed neural network ensembles according to sections 3 and 4. The networks are trained and we use the trained ensembles of both the proposed models to make predictions on the test phase time-series.

Step 2. Prediction on test-phase time-series: In this step, we apply the trained models to make predictions on the test phase Sunspot time-series. Figures 7 and 8 illustrate the predictions made by the first-order rule based NN model and the fuzzy rule based NN model respectively on the test phase sunspot series. In order to quantify the prediction accuracy, we use three well-known error metrics, i.e., the mean square error (MSE), the root mean square error (RMSE) and the normalized mean square error (NMSE). Let $c_{test}(t)$ denote the value of the test-period time-series at the time-instant t and let $c'(t)$ be the predicted time-series value for the same time-instant. The above mentioned error metrics can be defined by the following equations:

$$MSE = \frac{\sum_{t=1}^{N} (c_{test}(t) - c'(t))^2}{N} \tag{17}$$

$$RMSE = \sqrt{\frac{\sum_{t=1}^{N} (c_{test}(t) - c'(t))^2}{N}} \tag{18}$$

$$NMSE = \frac{\sum_{t=1}^{N} (c_{test}(t) - c'(t))^2}{\sum_{t=1}^{N} (c_{test}(t) - \bar{c}(t))^2} \tag{19}$$

where N is the total number of data points in the test phase time-series and $\bar{c}(t)$ is the average of all N data points. In Table 3, we present a comparison in the prediction performance of the proposed approaches with respect to other existing neural models in the literature.

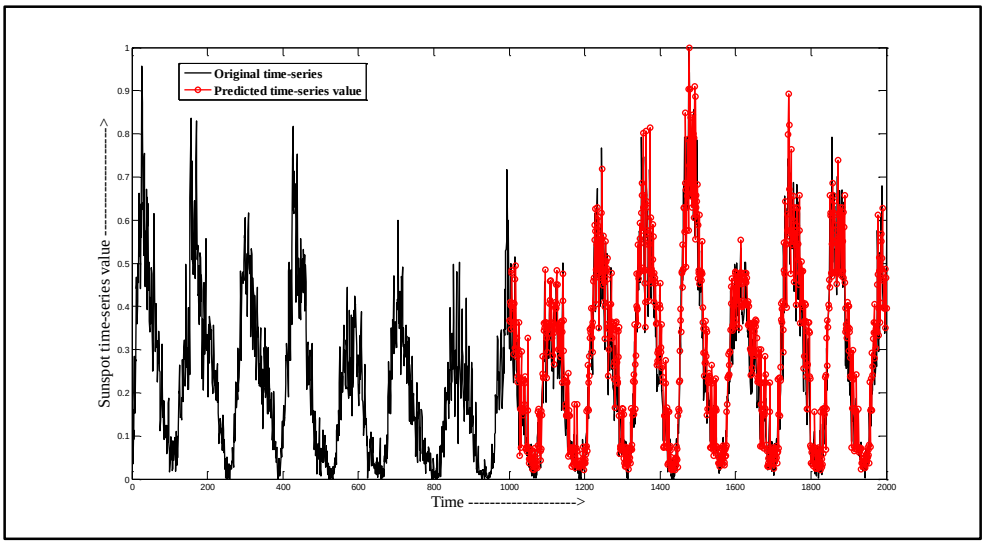

Fig. 7. Prediction of sunspot time-series using first-order transition rule based neural network model

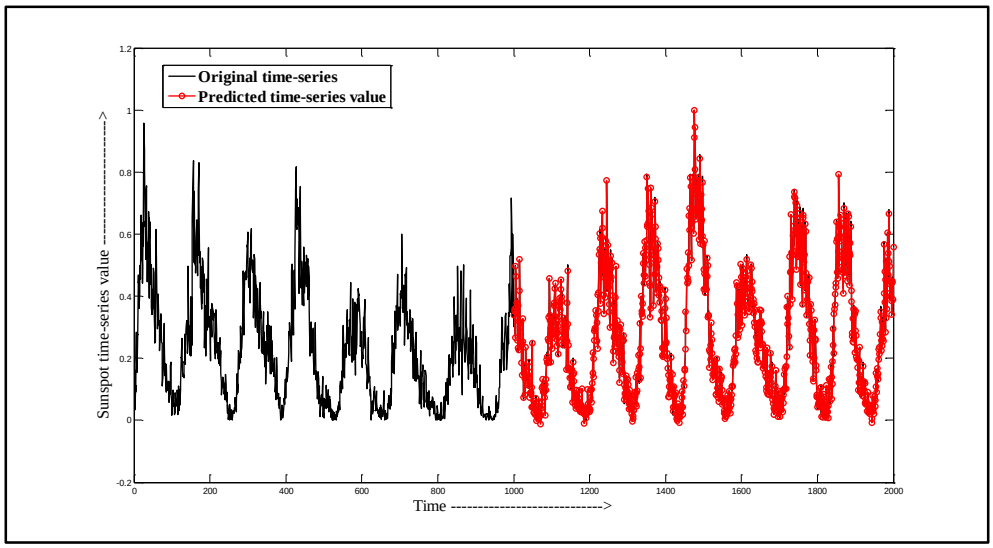

Fig. 8. Prediction of sunspot time-series using fuzzy rule based neural network model

Table 3
Comparison of prediction errors of existing methods in the literature with the proposed methods on the Sunspot time-series

Model	Prediction Error		
	MSE	RMSE	NMSE
Koskela *et al*. [39]			9.79E-02
Ma *et al*. [38]		1.29E-02	2.80E-03
Smith *et al.* [17]	2.32E-04	1.52E-02	9.46E-04
Ardalani-Farsa*et al.* [18]	1.4078E-4	0.0119	5.9041E-04
Proposed method 1	**0.843E-04**	**0.0071**	**4.09E-04**
Proposed method 2	**0.094E-04**	**8.7472E-04**	**3.0315E-04**

It is evident from Table 3 that the proposed models outperform other competitive models existing in the literature by a relatively large margin. Furthermore, among the two proposed methods, the second method which uses fuzzified first-order transition rules for training, leads to comparatively better prediction performance. This is primarily because of the fact that fuzzy representation of time-series data points lowers approximation errors in comparison to discrete quantization of data points with respect to respective partition mid-points. It can thus be concluded that the proposed models with their high prediction accuracy can be efficiently utilized for prediction of chaotic time-series.

5.1 Experiment 2: TAIEX close-price prediction

In this experiment we apply the proposed models for prediction of the TAIEX close price time-series for the period 1990 to 2004. For each year, the time-series is divided into two periods: i) the training period from January to October and ii) the testing period from November to December. The following steps are carried out for the experiments:

Step 1. Training Phase: Partitioning, First-order rule extraction and neural network training. Following sections 3 and 4, the training period time-series is first partitioned. For partitioning, we have experimentally chosen 40 equi-spaced partitions as that yields the best prediction results. The first-order transition rules thus extracted from the time-series for each year are then segregated into training sets for the neural networks and subsequently modified into mid-point to mid-point mappings for the discrete rule based model and into membership values to mid-point mappings for the fuzzy rule based model. The neural networks are trained using the back propagation algorithm on the training sets obtained above.

Step 2. Testing Phase: Prediction on test series. In the test phase, the trained neural networks of both the proposed models are used to make predictions on the time-series of the testing period for each year (1990-2004). In order to measure the prediction error of the models, we use the RMSE error metric as defined in equation (18). A comparative study of the proposed models with various existing models in the literature has been performed. For the period, 1990 to 1999, the proposed models are compared with conventional models [23], weighted models [23], Chen and Chen's

model [27], Chen *et. al*'s model [26], Chen and Kao's model [28] and Cai *et. al*'s model [29]. The results of all the above mentioned models are obtained from [29] and are shown in Table 4. For the period 1999-2004, the proposed models are compared with methods specified in [25]-[34] and the results are summarized in Table 5. The minimum RMSE values for each year have been shown in bold. Figures 9 and 10 illustrate the variations in the RMSE values graphically over the years 1990-1999 and 1999-2004 respectively, for the proposed models along with the next best model (Cai *et. al* [29]).

Table 4
Comparison of proposed models with existing methods for the period 1990-1999

Years / Models	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	Average
1.Conventional models [23]	220	80	60	110	112	79	54	148	167	149	117.9
2.Weighted models [23]	227	61	67	105	135	70	54	133	151	142	114.5
3.Chen and Chen [27]	172.89	72.87	43.44	103.21	78.63	66.66	59.75	139.68	124.44	115.47	97.70
4. Chen *et. al* [26]	174.62	43.22	42.66	104.17	94.6	54.24	50.5	138.51	117.87	101.33	92.17
5. Chen and Kao [28]	**156.47**	56.50	36.45	126.45	62.57	105.52	51.50	125.33	**104.12**	87.63	91.25
6.Cai *et. al* [29]	187.10	39.58	39.37	101.80	76.32	56.05	**49.45**	123.98	118.41	102.34	89.44
7.Proposed method 1	163.13	**33.88**	40.63	103.94	78.43	62.75	49.62	126.21	106.34	91.27	85.62
8.Proposed method 2	158.26	36.41	**34.38**	**98.73**	**61.14**	**50.48**	51.16	**121.67**	112.44	**87.03**	**81.17**

Table 5
Comparison of proposed models with existing methods for the period 1999-2004

Years / Methods	1999	2000	2001	2002	2003	2004	Average
1.Huarng *et al.*[25](Using NASDAQ)	NA	158.7	136.49	95.15	65.51	73.57	105.88
2.Huarng *et al.*[25](Using Dow Jones)	NA	165.8	138.25	93.73	72.95	73.49	108.84
3.Huarng *et al.*[25](Using M_{1b})	NA	160.19	133.26	97.1	75.23	82.01	111.36
4.Huarng *et al.*[25](Using NASDAQ and M_{1b})	NA	157.64	131.98	93.48	65.51	73.49	104.42
5.Huarng *et al.*[25](Using Dow Jones and M_{1b})	NA	155.51	128.44	97.15	70.76	73.48	105.07
6.Huarng *et al.*[25] (Using NASDAQ, Dow Jones and M_{1b})	NA	154.42	124.02	95.73	70.76	72.35	103.46
7.Chen *et al.*[30],[31],[32]	120	176	148	101	74	84	117.4
8.U_R Model [31],[32]	164	420	1070	116	329	146	374.2
9.U_NN Model [31],[32]	107	309	259	78	57	60	145.0
10.U_NN_FTS Model [31] [32],[33]	109	255	130	84	56	116	125.0
11.U_NN_FTS_S Model [31],[32],[33]	109	152	130	84	56	116	107.8
12.B_R Model [31],[32]	103	154	120	77	54	85	98.8
13.B_NN Model [31],[32]	112	274	131	69	52	61	116.4
14.B_NN_FTS Model [31],[32]	108	259	133	85	58	67	118.3
15.B_NN_FTS_S Model [31],[32]	112	131	130	80	58	67	96.4
16.Chen and Chen.[27] (Using Dow Jones)	115.47	127.51	121.98	74.65	66.02	58.89	94.09
17.Chen and Chen [27] (Using NASDAQ)	119.32	129.87	123.12	71.01	65.14	61.94	95.07
18.Chen and Chen [27] (Using M_{1b})	120.01	129.87	117.61	85.85	63.1	67.29	97.29
19.Chen and Chen [27] (Using NASDAQ and Dow Jones)	116.64	123.62	123.85	71.98	58.06	57.73	91.98
20.Chen and Chen [27] (Using Dow Jones and M_{1b})	116.59	127.71	115.33	77.96	60.32	65.86	93.96
21.Chen and Chen [27] (Using NASDAQ and M_{1b})	114.87	128.37	123.15	74.05	67.83	65.09	95.56
22.Chen and Chen [27] (Using NASDAQ, Dow Jones and M_{1b})	112.47	131.04	117.86	77.38	60.65	65.09	94.08
23. Karnik-Mendel [34] induced stock prediction	116.60	128.46	120.62	78.60	66.80	68.48	96.59
24. Chen *et al.* [26] (Using NASDAQ, Dow Jones and M_{1b})	101.47	122.88	114.47	67.17	52.49	52.84	85.22
25. Chen and Kao [28]	87.67	125.34	114.57	76.86	54.29	58.17	86.14
26.Cai *et al.* [29]	102.22	131.53	112.59	60.33	51.54	**50.33**	84.75
27. Proposed method 1	91.27	120.16	108.63	**59.18**	46.88	91.62	86.29
28. Proposed method 2	**87.03**	**102.04**	**99.49**	61.25	**39.14**	70.11	**76.51**

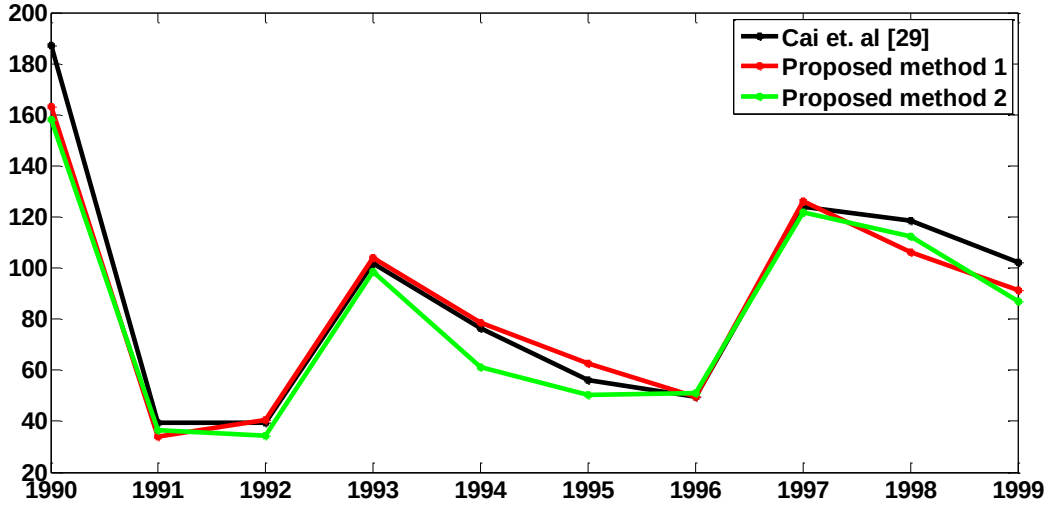

Fig. 8. RMSE Error values for Cai et.al [29] and the two proposed models for the years 1990-1999

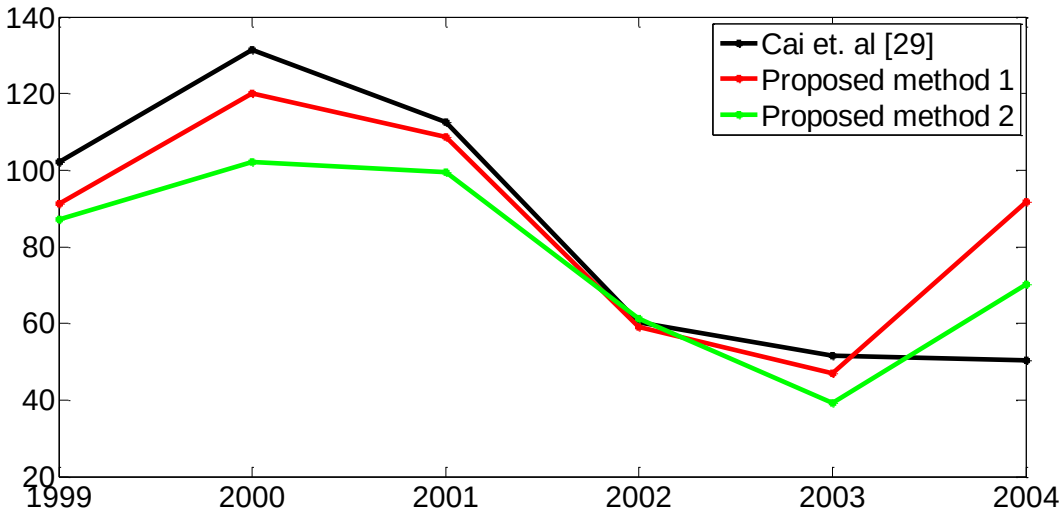

Fig. 9. RMSE Error values for Cai et.al [29] and the two proposed models for the years 1990-2004

As is evident from Tables 4 and 5, the proposed models clearly outperform the other existing models in the literature. The fuzzy rule-based neural network model outperforms the next best existing model by 9.25% for the period 1990-1999 and by 9.72% for the period 1999-2004. The results indicate that the proposed models can be effectively utilized in financial prediction.

6. Conclusion

In this paper, we have presented a novel grouping scheme of first-order transition rules obtained from a partitioned time-series for the purpose of fuzzy-induced neural network based prediction. In this direction, we have proposed two models. The first model uses first-order transition rules segregated into groups representing injective mappings from antecedents to consequents of a rule. Each rule in a group thus obtained possesses a distinct antecedent and each such group is used to train a separate neural network in the ensemble. This helps in realizing the simultaneous and concurrent firing of multiple rules during prediction. Furthermore, the individual predictions of the networks are weighted according to the probability of occurrence of their corresponding transition rules and the weighted sum thus obtained is treated as the final predicted value of the model. This helps in taking the recurrence of transition rules into account while making forecasts.

The second proposed model modifies and extends the training scheme of the first, by considering each partition of the time-series as a fuzzy set to identify the membership of a data point in its respective partition. The advantage of such an approach lies in utilizing the inherent fuzziness involved in identifying the partition to which a time-series data point belongs, thereby reducing the approximation error induced due to quantization of a time-series data point with respect to its partition mid-point value. The first-order transition rules are thus converted into fuzzy first-order rules and segregated into training sets following the grouping scheme as mentioned above.

Extensive experiments carried out on real life chaotic time-series like the sunspot [36] as well as on economic time-series like the TAIEX [37] reveal a high prediction accuracy for both the proposed models. Furthermore, it is also observed that the performance of the fuzzy rule based neural network ensemble is comparatively better than its predecessor in both the experiments carried out. Prediction performance can possibly be further increased by approaches like optimized partitioning, higher order transition rules for training and chronological weight assignment of transition rules. Such approaches form a future scope for the work. Thus, with the high forecast accuracy and low prediction error of the proposed models compared to other existing models in the literature, we can conclude that the said models can be effectively utilized for real-life time-series forecasting applications.

References

[1] S.M. Chen, and J.R. Hwang, "Temperature Prediction Using Fuzzy Time Series," *IEEE Trans. Syst., Man, Cybern., Part-B*, vol. 30, no. 2, pp. 263-275, Apr. 2000.

[2] C.L. Wu, and K.W. Chau, "Prediction of Rainfall Time Series Using Modular Soft Computing Methods," *Elsevier, Engineering Applications of Artificial Intelligence*, vol. 26, pp. 997-1007, 2013.

[3] A. Morales-Esteban, F. Martinez-Alvarez, A. Troncoso, J.L. Justo, and C. Rubio-Escudero, "Pattern Recognition to Forecast Seismic Time Series," *Elsevier, Expert Systems with Applications*, vol. 37, pp. 8333-8342, 2010.
[4] O. Barnea, A. R. Solow, and L. Stone, *"*On Fitting a Model to a Population Time Series with Missing Values,*" Israel Journal of Ecology and Evolution*, vol. 52, pp. 1-10, 2006.
[5] A. Jalil, and M. Idrees, *"*Modeling The Impact Of Education On The Economic Growth: Evidence From Aggregated And Disaggregated Time Series Data Of Pakistan,*" Elsevier, Econ. Model*, vol. 31, pp. 383-388, 2013.
[6] G. E. P. Box and G. Jenkins, *Time Series Analysis, Forecasting and Control*. San Francisco, CA: Holden-Day, 1976.
[7] C. C. Wang, "A Comparison Study Between Fuzzy Time Series Model and ARIMA Model for Forecasting Taiwan Export," *Elsevier, Expert Systems with Applications*, vol. 38, no. 8, pp. 9296-9304, 2011.
[8] B. R. Chang, and H. F. Tsai, "Forecast Approach Using Neural Network Adaptation to Support Vector Regression Grey Model and Generalized Auto-Regressive Conditional Heteroscedasticity," *Elsevier, Expert Systems with Applications*, vol. 34, no. 2, pp. 925-934, 2008.
[9] C. P. Tsokos, "K-th Moving, Weighted and Exponential Moving Average for Time Series Forecasting Models," *European Journal of Pure and Applied Mathematics*, vol. 3, no. 3, pp. 406-416, 2010.
[10] G. P. Zhang, "Time Series Forecasting using a Hybrid ARIMA and Neural Network Model," *Elsevier, Neurocomputing*, vol. 50, pp. 159-175, 2003.
[11] L. A. Zadeh, "Fuzzy Sets," *Information and Control*, vol. 8, pp. 338-353, 1965.
[12] G. P. Zhang, "Neural Networks for Classification: A Survey," *IEEE Trans. Sys., Man, Cybern. Part C (Applications and Reviews)*, vol. 30, no. 4, pp. 451-462, 2000.
[13] L. Ma, and K. Khorasani, "New Training Strategies for Constructive Neural Networks with Application to Regression Problems," *Elsevier, Neural Networks*, vol. 17, no. 4, pp. 589-609, 2004.
[14] T. Hill, M. O' Connor, and W. Remus, "Neural Network Models for Time Series Forecasts," *Management Science*, vol. 42, no. 7, pp. 1082-1092, 1996.
[15] C. Hamzacebi, "Improving Artificial Neural Networks' Performance in Seasonal Time Series Forecasting," *Elsevier, Information Sciences*, vol. 178, pp. 4550-4559, 2008.
[16] D. T. Mirikitani, and N. Nikolaev, "Recursive Bayesian Recurrent Neural Networks for Time-Series Modeling," *IEEE Trans. Neural Networks*, vol. 21, no. 2, pp. 262-274, 2010.
[17] C. Smith, and Y. Jin, "Evolutionary Multi-Objective Generation of Recurrent Neural Network Ensembles for Time Series Prediction," *Elsevier, Neurocomputing*, vol. 143, pp. 302-311, 2014.
[18] M. Ardalani-Farsa, and S. Zolfaghari, "Chaotic Time Series Prediction with Residual Analysis Method Using Hybrid Elman-NARX Neural Networks," *Elsevier, Neurocomputing*, vol. 73, pp. 2540-2553, 2010.
[19] F. Gaxiola, P. Melin, F. Valdez, and O. Castillo, "Interval Type-2 Fuzzy Weight Adjustment for Backpropagation Neural Networks with Application in Time Series Prediction," *Elsevier, Information Sciences*, vol. 260, pp. 1-14, 2014.
[20] Q. Song, B.S. Chissom, Fuzzy time series and its model, Fuzzy Sets Syst. 54 (3), (1993) 269–277.
[21] Q. Song, B.S. Chissom, Forecasting enrollments with fuzzy time series—part I, Fuzzy Sets Syst. 54 (1) (1993) 1–9.
[22] Q. Song, B.S. Chissom, Forecasting enrollments with fuzzy time series—part II, Fuzzy Sets Syst 62 (1) (1994) 1–8.
[23] H.K. Yu, Weighted fuzzy time series models for TAIEX forecasting, Physica A, 349 (2005) 609–624.
[24] Mu-Yen Chen, "A High-Order Fuzzy Time Series Forecasting Model for Internet Stock Trading," *Elsevier, Future Generation Computer Systems*, vol. 37, pp. 461-467, 2014.
[25] K. Huarng, H.K. Yu, Y.W. Hsu, A multivariate heuristic model for fuzzy time-series forecasting, IEEE Trans. Syst. Man Cybern. B Cybern. 37 (4) (2007) 836–846.
[26] S. M. Chen, H. P. Chu, and T. W. Sheu, "TAIEX Forecasting Using Fuzzy Time Series and Automatically Generated Weights of Multiple Factors," *IEEE Trans. Sys., Man, Cybern., Part A: Systems and Humans*, vol. 42, no. 6, pp. 1485-1495, 2012.
[27] S.M. Chen, C.D. Chen, TAIEX forecasting based on fuzzy time series and fuzzy variation groups, IEEE Trans. Fuzzy Syst. 19 (1) (2011) 1–12.
[28] S.M. Chen, P.Y. Kao, TAIEX forecasting based on fuzzy time series, particle swarm optimization techniques and support vector machines, Inform. Sci. 247 (2013) 62–71.
[29] Q. Cai, D. Zhang, W. Zheng, and S. C. H. Leung, "A new fuzzy time series forecasting model combined with ant colony optimization and auto-regression," Knowledge-Based Systems 74(2015) 61-68.
[30] S. M. Chen, "Forecasting enrollments based on fuzzy time series," Fuzzy Sets Syst., vol. 81, no. 3, pp. 311–319, Aug. 1996.
[31] T. H. K. Yu, K. H. Huarng, "A bivariate fuzzy time series model to forecast the TAIEX," Expert Syst. Appl., vol. 34, no. 4, pp. 2945–2952, 2008.
[32] T. H. K. Yu and K. H. Huarng, "Corrigendum to "A bivariate fuzzy time series model to forecast the TAIEX," Expert Syst. Appl., vol. 37, no. 7, p. 5529, 2010.
[33] K. Huarng and T. H. K. Yu, "The application of neural networks to forecast fuzzy time series," Phys. A, vol. 363, no. 2, pp. 481–491, May 2006.
[34] N. N. Karnik, J. M. Mendel, "Applications of type-2 fuzzy logic systems to forecasting of time-series", Elsevier, Information Sciences 120, (1999) 89-111.
[35] S. Elanayar V. T, and Y. C. Shin, "Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems," *IEEE Trans. on Neural Networks*, vol. 5, no. 4, pp. 594-603, 1994.
[36] Y.R. Park, T.J. Murray, C. Chen, Predicting sun spots using a layered perceptron neural network, IEEE Transactions on Neural Networks 1 (2) (1996) 501–505.
[37] TAIEX.[Online].Available: http://www.twse.com.tw/en/products/indices/tsec/taiex.php.
[38] Q. Ma, Q. Zheng, H. Peng, T. Zhong, L. Xu, Chaotic time series prediction based on evolving recurrent neural networks, in: Proceedings of the Sixth Interna- tional Conference on Machine Learning and Cybernetics, Hong Kong, 2007.
[39] T. Koskela, M. Lehtokangas, J. Saarinen, K. Kaski, Time series prediction with multilayer perceptron, FIR and Elman neural networks, in: Proceedings of the World Congress on Neural Networks, 1996, pp. 491–496.