A Meta-cognitive Recurrent Fuzzy Inference System with Memory Neurons (McRFIS-MN) and its Fast Learning Algorithm for Time Series Forecasting

Subhrajit Samanta
ERI@N-Interdisciplinary Graduate School
Nanyang Technological University
Singapore
sama0021@e.ntu.edu.sg

Shubhangi Ghosh
Electrical Engineering Department
Indian Institute of Technology Madras
Chennai, India
ee15b129@smail.iitm.ac.in

Suresh Sundaram
Associate Professor, SCSE
Nanyang Technological University
Singapore
ssundaram@ntu.edu.sg

***Abstract*—In this paper, a Meta-cognitive Recurrent Fuzzy Inference System is proposed where recurrence is brought using Memory type Neurons (McRFIS-MN) to retain the effect of all past instances, while the meta-cognition component is employed to control the learning process, by deciding what-to-learn, when-to-learn and how-to-learn from the training data. The McRFIS-MN model has five layers, and Memory Neurons (MN) are employed only in the layers handling crisp values. The antecedent parameters are set randomly while only the consequent weights of the network are updated using a one-shot type projection based learning algorithm through time (PBLT) which makes the learning very fast. The performance evaluation of McRFIS-MN has been carried out using benchmark problems in the areas of nonlinear system identification and time-series forecasting. The results are evaluated against some of the most popular neural fuzzy methods and the obtained results indicate that McRFIS-MN performs better in terms of speed while achieving better or similar accuracy.**

***Index Terms*—Meta-cognition, Neural Fuzzy Inference System, Memory Neuron, Projection Based Learning through Time, Time Series Forecasting, Dynamic System Identification**

I. Introduction

Time series forecasting is a sub-field of predictive analytics which involves developing models to fit historical data in order to perform future predictions. Time series forecasting has numerous useful applications such as energy requirement prediction [1], weather forecasting [2] and financial indicator prediction [3] etc. In a time series, past observations have an inherent temporal ordering, which makes the problem of forecasting more difficult.

Classical approaches such as Non-linear Auto-Regressive Models (NARMA) [4] have been used to approximate relationships between future states, past states, and exogenous inputs. However, NARMA requires the general functional relationship to be specified and thus is inadequate to capture arbitrary functions mapping from past states to future. To overcome this, Artificial Neural Networks (ANNs) [5] were introduced. ANNs can learn any arbitrary mapping owing to the universal approximation theorem [6]. Static feed-forward ANNs can't capture dynamic relationships between system outputs and inputs well, hence to model the dynamic relationship recurrence was introduced in artificial neural networks. Recurrence can be modeled in two broad ways - using the tapped delay approach or using a feedback system. Tapped delay systems explicitly feed in the past outputs as input at every time-step and thereby assume the order of the system to be known and fixed. The other way is to use feedback systems which can further be classified into local recurrence systems and global recurrence systems based on the type of feedback. In local recurrence [7], there is self-feedback in neurons and a feedback is restricted to a single layer. In global recurrence [8], delayed information from one layer is fed to another layer. Memory neurons is another popular way of bringing in recurrence, which falls in the third class of both local and global type recurrence. In Memory Neural Network (MNN) [9], a memory neuron is associated with each network neuron whose scalar output summarizes the history of past activation of that unit. Thus, a memory neural network can model the dynamic trends of a time series more effectively.

To model the uncertainty in practical time series, researchers have resorted to fuzzy inference systems [10], for its human-like linguistic behavior. The functional equivalence between a Radial Basis Function Neural Network (RBFNN) and a Gaussian Fuzzy Inference System (FIS), has been used to determine the structure i.e. the number of fuzzy rules, the initial positions (centers) of the rule membership functions and other parameters of the FIS. Fixed structure FIS are not usually feasible because a large number of hyper-parameters including the number of rules have to be set heuristically and the optimal structure may vary depending on the nature and size of the data. Self-adaptive fuzzy inference systems [11]–[13] start with zero rules and build up the required number of rules as they run through the training data. Novelty of a sample is determined based on the corresponding prediction error. This approach doesn't rely on availability of extra knowledge

Shubhangi Ghosh was working as a student intern under supervision of Prof. S Sundaram at SCSE, NTU, Singapore and was funded by NTU-India Connect Programme, while working on this project and writing this paper.

about the input data unlike data-querying or active learning approaches [14]. If an incoming sample is very novel, a new fuzzy rule is added or else the existing rules are updated based on the distance of the sample from the existing fuzzy rule centers. The distance measure here is estimated using a spherical potential formulation, which has been described in detail in later sections. There are provisions for samples to be deleted or kept in reserve as well as rule pruning. Type-II self-adaptive fuzzy inference systems [15], [16] are more effective in online feature selection and capturing temporal behaviour, and thus can obtain greater accuracy with fewer rules. But as Type-II FIS makes the neuron structure heaver, prediction time is increased and hence Type-II FIS neurons have been avoided. Time series forecasting problems can be handled both by batch learning scheme or sequential learning scheme. In batch learning, the complete training data is assumed to be available before training commences, whereas in sequential learning data instances arrive one-by-one and past samples are not revisited. In batch learning , whenever new data arrives the network has to be retrained all over again. Since in practical applications, data usually arrives sequentially, batch learning may sometimes be infeasible. The proposed model can be used both in a batch or a sequential learning scenario. In a batch learning scenario, multiple epochs are run over the training data and the samples kept in reserve (after one epoch) are given a chance to alter the network in the subsequent epochs (or that epoch itself). In a sequential learning scenario, since the complete training data is not available, each training instance is visited only once. In this work, although the entire training data was available beforehand for all the experiments (as is the case for batch learning), we train our model in a sequential manner by considering one sample at a time.

For updating the existing parameters, Projection-Based Learning algorithm through Time (PBLT) has been used. It is an one-shot learning algorithm which is fast, thus making it well suited for any kind of learning scenario (both batch and sequential). The conventional Backpropagation Through Time (BPTT) [17] algorithm is fairly accurate when it comes to handling dynamic systems, however, the convergence speed is inherently slow due to infinitesimally small gradients near the optima. Moreover, an improper choice of hyper-parameters such as learning rate etc. may lead to problems of stability and very slow convergence due to a large number of update iterations. The Extended Kalman Filter (EKF) based learning algorithm [18] converges in much fewer iterations than BPTT, as it linearizes the state (output) transition relationship at each time-step and gives an approximate estimate of the parameters which minimize the squared error.To speed up the learning process further, Projection-Based Learning algorithm through Time (PBLT) [19] is used. PBLT considers the loss function to be the sum of squared hinge loss and expresses it exclusively as a function of the weights connecting to the output layer, assuming all other dependencies of the loss to be constant with respect to variations in input. This leads to the existence of a closed form solution for minimization of the loss function. Thus, the output weights can be estimated in one shot making the learning process very fast.

In section II, problem definition and design aspects of the McRFIS-MN model will be discussed followed by the meta-cognitive learning process. In section III the performance evaluation, comparison of results and analysis will be presented followed by the conclusion.

II. Meta-cognitive Recurrent Fuzzy Inference System (McRFIS-MN)

In general, the problem of a time series forecasting can be stated as to predict the τ time steps ahead value $y(k+\tau)$ based on its past output values $y(k)$ and exogenous inputs $\mathbf{u}(k)$(i.e. $\mathbf{u}(k) = [u_1(k), u_2(k), \cdots, u_P(k)]^T \in \Re^P$) as given below,

$$y(k+\tau) = h\big(y(k), \cdots, y(k-n_y); \mathbf{u}(k), \cdots, \mathbf{u}(k-n_u)\big) \quad (1)$$

where $h(.)$ is an arbitrary nonlinear function representing the system and n_y, n_u are the required number of lags for the time series prediction. On the other hand, while treating a nonlinear dynamic system identification problem, the aim is to approximate closely the input-output functional relationship of the dynamical system. The training data, $[y(k), \mathbf{u}(k)]^T \in \Re^{\hat{m}}$ (where $\hat{P} = P + 1$) over time provides the necessary information on inferring the dynamical relationship present in the forecasting problem.

In this paper, memory neurons [9] are adopted into the fuzzy-inference architecture of McFIS [12] to bring recurrence in a cellular level while the meta-cognitive part of the fuzzy inference system evolves and determines the optimum structure (i.e. number of fuzzy rules).

Description of Memory Neuron (MN) [9] : Memory neurons contain one regular neuron which accounts for the static mapping of the corresponding inputs whereas it has one memory output as well which represents the dynamic mapping of the inputs as shown in the equations below. If the input to the network neuron is $\mathbf{d}^I(k) \in \Re^p$, any p dimensional vector, then the network neuron output ($d_O(k)$) and memory neuron output ($d_O^m(k)$) can be given by,

$$d_O(k) = f\left(\mathbf{d}^I(k)\right) \quad (2)$$

$$d_O^m(k) = \alpha d_O(k) + (1-\alpha) d_O^m(k-1) \quad (3)$$

where $f(.)$ can be any nonlinear function depending on the layer the memory neuron is in and α is the memory neuron parameter ($0 \leq \alpha \leq 1$). It is apparent from 3 that the memory output will retain the effect of all past instances (although with exponentially decreasing values). These memory neurons are then incorporated in the crisp layers of the proposed neuro-fuzzy inference system to introduce recurrence at the cellular(building block) level into the network.

Using McRFIS-MN the τ-time step ahead prediction problem gets reduced to determine the $\hat{h}$ in,

$$\hat{y}(k+\tau) = \hat{h}\big(y(k), \mathbf{u}(k), \mathbf{w}\big) \quad (4)$$

where $\mathbf{w}$ represents McRFIS-MN's weight parameters which are to be determined during training. Note that, the input to the McRFIS-MN is $y(k)$ and $\mathbf{u}(k)$ and the output is $\hat{y}\,(k+\tau)$.

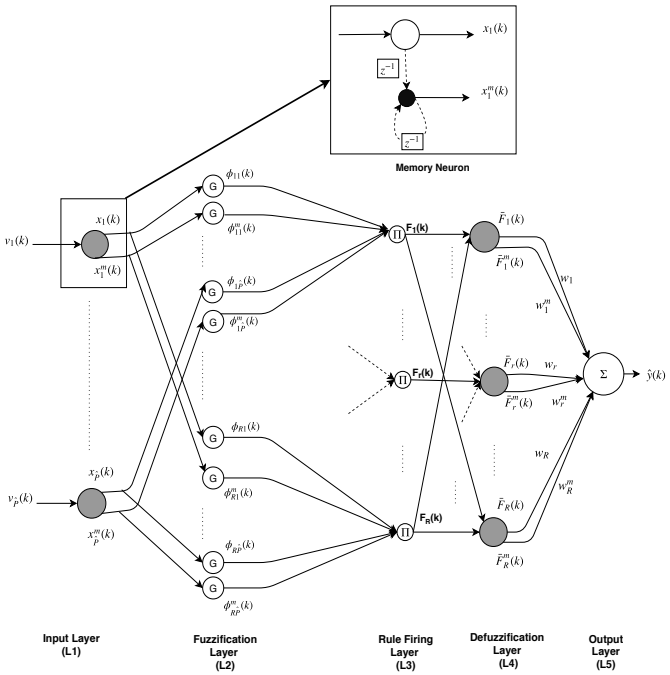

Figure 1: Network architecture of McRFIS-MN

Unlike other networks McRFIS-MN does not need the past lags of $y(k)$ and $\mathbf{u}(k)$ as separate inputs since MNs take care of that.

A. Structure of McRFIS-MN

The architecture of the proposed evolving neuro-fuzzy inference system with the memory neurons is shown in Figure 1. It consists of a five-layer neuro-fuzzy network. Here, the first layer is the input layer, the second layer is the fuzzification layer, the third layer is the rule firing layer, the fourth layer is the defuzzification layer and the last layer is the output layer. McRFIS-MN incorporates memory neurons (represented by solid circles in the figure 1) in input and defuzzification layers. The input layer has $\hat{P}$ memory neurons. The fuzzification layer and Gaussian rule layer has regular fuzzy neural nodes, whereas the defuzzification layer has R memory neurons corresponding to the R number of rules followed by the final output layer.

Input Layer: The input layer has $\hat{P}$ input memory neurons which receive the P input features $\mathbf{u}(k)$ and one lagged output $y(k)$, hence number of nodes is $\hat{P} = P + 1$. The augmented input vector is represented as $\mathbf{v}(k) = [y(k), \mathbf{u}(k)]^T$. All of the neurons in this layer are memory neurons. For the i^{th} memory neuron in the input layer (layer designated by $'I'$) the input is $\mathbf{v}_i(k)$ and the corresponding outputs can be defined as below,

$$x_i(k) = f_i^I\big(\mathbf{v}_i(k)\big) = \mathbf{v}_i(k), \quad i = 1, 2, \cdots, \hat{P} \tag{5}$$

$$x_i^m(k) = \alpha_i^I x_i(k) + (1 - \alpha_i^I) x_i^m(k-1) \tag{6}$$

where α_i^I is the memory neuron parameter ($0 \leq \alpha_i^I \leq 1$).

Fuzzification Layer: An initial arbitrary number of rules is heuristically set. This arbitrary value is one of the tunable hyperparameters. This number may be modified during the training process in accordance with the rule addition or rule pruning criteria. Each of the crisp outputs from the input layer MNs, i.e. $x_i(k)$ and $x_i^m(k)$, are transformed into fuzzy variables using fuzzy membership functions. Here, Gaussian membership functions are used to turn them into linguistic variables. The r^{th} (where $r = 1, 2, \cdots, R$) membership value for $x_i(k)$ is,

$$\phi_{ri}(k) = exp\left[-\frac{1}{2}\left[\frac{(x_i(k) - \mu_{ri})}{\sigma_r}\right]^2\right] \tag{7}$$

where μ_{ri} and μ_{ri}^m are the centers of the r^{th} membership functions for $x_i(k)$ and $x_i^m(k)$ respectively and σ_r is the width. So the membership value for $x_i^m(k)$ corresponding to r^{th} rule is given by,

$$\phi_{ri}^m(k) = exp\left[-\frac{1}{2}\left[\frac{(x_i^m(k) - \mu_{ri}^m)}{\sigma_r}\right]^2\right] \tag{8}$$

Rule Firing Layer: The R regular neurons in this layer do the rule aggregation of the fuzzy inference system using a t-norm operation which is realized by multiplying all the fuzzified input values ($\phi_{ri}(k), \phi_{ri}^m(k)$ where $r = 1, 2, \cdots, R$ and $i = 1, 2, \cdots, \hat{P}$) for each rules. So the firing strength corresponding to the r^{th} rule is,

$$F_r(k) = \Pi_{i=1}^{\hat{P}}\left(\phi_{ri}(k).\phi_{ri}^m(k)\right) \tag{9}$$

Defuzzification Layer : In the consequent part of the architecture, defuzzification layer has R number of memory neurons. So for the r^{th} memory neuron in this layer (layer designated by $'R'$) the input is $[F_1(k), F_2(k), \cdots, F_R(k)]^T$. Hence the corresponding output variables can be defined using eqn. 10 and eqn. 11 as below,

$$\bar{F}_r(k) = f_r^R\big(F_1(k), \cdots, F_R(k)\big) = \frac{F_r(k)}{\Sigma_{r=1}^R F_r(k)} \tag{10}$$

$$\bar{F}_r^m(k) = \alpha_r^R \bar{F}_r(k) + (1 - \alpha_r^R)\bar{F}_r^m(k-1) \tag{11}$$

where α_r^R is the memory neuron parameter ($0 \leq \alpha_r^R \leq 1$).

Output Layer: The final output node performs a weighted summation of the outputs from the memory neurons in the previous layer. The output from this layer is the τ step ahead prediction $y(k + \tau)$ is given as,

$$\hat{y}(k + \tau) = \Sigma_{r=1}^R w_r.\bar{F}_r(k) + \Sigma_{r=1}^R w_r^m.\bar{F}_r^m(k) \tag{12}$$

The rule parameters (μ and σ), the memory neuron parameters (α) in the input layer and the defuzzification layer and the output layer weights are all initialized to random values.

B. Meta-cognitive component of McRFIS-MN

The meta-cognition component of this network is a self-regulatory mechanism which works without any outside intervention to form an evolving optimized structure by deciding what-to-learn, when-to-learn, how-to-learn etc. Hence when a new sample arrives depending upon certain criteria one of the aforementioned strategies is employed.

1) Sample Deletion: With each incoming sample the corresponding prediction value is computed from McRFIS-MN followed by the prediction error, i.e. $e(k) = y(k) - \hat{y}(k)$ ($y(k)$,$\hat{y}(k)$ are the actual and predicted out for k^{th} sample). Now the strategy can be defined as below,

If $|e(k)| < E_d$ then delete k^{th} sample

Sample deletion strategy ensures that the network is not overfitted with training samples by not allowing McRFIS-MN to learn from similar samples. With this criteria, meta-cognition decides on what-to-learn.

2) Sample Learning: When the prediction error corresponding to a sample is greater than the delete threshold (i.e. $|e(k)| > E_d$) that implies the particular sample is knowledge rich and should be learned from. The learning can take place in two ways, either by adding a new fuzzy rule to the existing rule-base to accommodate a very novel sample or by updating the existing consequent weights of the McRFIS-MN for a knowledgeable but not so novel sample. This part of the meta-cognition is responsible for the how-to-learn strategies.

Rule Addition: When the sample is significantly knowledge rich then there might be a need for the addition of a new fuzzy rule to accommodate the knowledge of this sample into the network. While other algorithms in literature mostly use error based criteria to measure the novelty of the sample, McRFIS-MN uses spherical potential method [12] in addition to determine if a new rule should be added to the existing fuzzy rule base.

Spherical potential can be defined as the direct measure of knowledge contained in a sample, expressed in terms of squared distance mapping in the projected hyperdimensional spherical space (as the Gaussian function is employed to project the input features into the hyperdimensional space). The corresponding spherical potential can be computed in terms of the expression below,

$$\Psi = -\frac{2}{R}\Sigma_{r=1}^{R}\Sigma_{i=1}^{\hat{P}}(\phi_{ri}(k).\phi_{ri}^{m}(k)) \tag{13}$$

Please note that a lower potential indicates higher novelty and the vice versa and E_s is the potential threshold. On the other hand, there is an adding threshold E_a and the prediction error has to be greater than this threshold to be considered for a rule addition. Hence the Rule Addition criteria can be written as,

If $|e(k)| > E_a$ and $\Psi < E_S$ then add a new rule

when a new rule is added, then corresponding center and width of the rule along with the forward path weights (to the output node) need to be initialized as well. The Gaussian parameters are set as follows,

$$\begin{aligned}
&\mu_{(R+1)i} = x_i(k) \\
&\mu_{(R+1)i}^{m} = x_i^{m}(k), \quad i = 1, 2, \cdots, \hat{P} \\
&\sigma_{R+1} = min||x(k) - \mu_r||, \quad r = 1, 2, \cdots, R
\end{aligned} \tag{14}$$

With the new centers and spread $\bar{F}_{R+1}(k+1)$ can be computed for the current sample. We assign $\bar{F}_{R+1}^{m}(k+1)$ with the memory of the nearest existing rule (i.e, $\bar{F}_{nr}^{m}$). $\bar{F}_{1\cdots R}$ and $\bar{F}_{1\cdots R}^{m}$ are also recomputed because the number of rules have changed. The new memory neuron's α is initialized randomly like before. Once $\bar{F}_{R+1}(k+1)$ and $\bar{F}_{R+1}^{m}(k+1)$ are computed they are allocated two new weights w_{R+1},w_{R+1}^{m}. These two weights are initialized so as to drive the prediction error at that instance to zero to fully exploit the localization property of the gaussian membership function. For completeness of the system of equations, the network neuron weight and the memory neuron weight are initialized proportionately to the corresponding weights of the nearest rule. The residue error is estimated as,

$$\hat{e}(k) = y(k+\tau) - (\Sigma_{r=1}^{R} w_r.\bar{F}_r(k) + \Sigma_{r=1}^{R} w_r^{m}.\bar{F}_r^{m}(k)) \tag{15}$$

The weights corresponding to the new rule are now initialised as,

$$\begin{aligned}
&w_{R+1}F_{R+1}(k) + w_{R+1}^{m}F_{R+1}^{m}(k) = \hat{e}(k) \\
&\frac{w_{R+1}}{w_{nr}} = \frac{w_{R+1}^{m}}{w_{nr}^{m}}
\end{aligned} \tag{16}$$

Projection Based Learning through Time (PBLT) for weights Update: When the instantaneous error is greater than the update threshold E_l but is not novel enough for rule addition then the knowledge of the sample is used to update the consequent weights of McRFIS-MN using a one-shot projection based learning through time (PBLT) [19].

If $|e(k)| \geq E_l$ then update the weights

A cost function is defined as the summation of squared error for all the the training samples (i.e, S) and the goal is to determine the weights **w** such that cost function is minimized.

$$\begin{aligned}
&J(\mathbf{w}) = \frac{1}{2}\Sigma_{k=1}^{S}e^2(k) \\
&\mathbf{w}^{\star} = argminJ(\mathbf{w})
\end{aligned} \tag{17}$$

The optimal $\mathbf{w}^{\star}$ is estimated such that the total energy reaches its minimum. Making partial derivative of $J(\mathbf{w})$ with respect to **w**, equating it to zero and rearranging we can have the following equation,

$$A\mathbf{w} = B \tag{18}$$

here **w** is the weight vector consisting weights w_r, w_r^{m} (i.e, $\mathbf{w}\epsilon\Re^{2Rx1}$), matrix A is consisted of elements a_{rr^*} (i.e, $A\epsilon\Re^{2Rx2R}$) and B is consisted of b_r such that $B\epsilon\Re^{2Rx1}$ where,

$$\begin{aligned}
&a_{rr^*} = \Sigma_{k=1}^{S}\hat{F}_r(k).\hat{F}_{r^*}(k) \\
&b_r = \Sigma_{k=1}^{S}\hat{F}_r(k).y(k), \quad r, r^* = 1, 2, \cdots, 2R
\end{aligned} \tag{19}$$

where $\hat{F} = [\bar{F}; \bar{F^m}]$ is the augmented vector ($\hat{F}\epsilon\Re^{2Rx1}$) made of all the normalized firing strengths (regular and memory both) and $y(k)$ is the actual output.

It is proved in [19] that second derivative of the cost function with respect to the weight vector is positive hence solution of Eqn. 20 is ensured to produce the optimal weight as,

$$\mathbf{w}^{\star} = A^{-1}.B \tag{20}$$

Rule Pruning: A particular fuzzy rule will be considered insignificant if for a certain number of consecutive samples

(say N_p) the contribution of the rule β_r is lower than pruning threshold E_p. A spurious sample (outlier) can pose as a novel knowledgeable sample and increase the number of rules. To ensure that such rules are not present in the optimal architecture rule pruning strategy is executed. Contribution of a particular rule is calculated as below,

$$\beta_r = \bar{F}_r(k).\bar{F^m}_r(k).|w_r.w_r^m.e(k)| \tag{21}$$

β_r is the product of the rule consequents and the rule weights, hence denotes the significance of the rule for the current input instance. If $\beta_r < E_p$ for N_p samples then the r^{th} rule is pruned

3) Sample Reserve: When a sample meets the delete condition it is discarded but if a sample contains knowledge (doesn't meet delete criteria) but doesn't also qualify any learning criteria then that sample is deferred to the rear end of the training and learned at a later stage. This strategy is termed as sample reserve strategy and is employed to ensure that a knowledgeable sample isn't discarded because the network didn't consider it worthy at that point in time. This takes care of the when-to-learn part of the metacognition.

III. Experimental Results

In recent years, researchers have mostly concentrated on deep neural structure whereas this work focuses on shallow neural fuzzy inferences system to reduce prediction time while obtaining competent prediction accuracy. Thus the performance of McRFIS-MN is only compared with state-of-the art shallow NFIS systems. In this section, McRFIS-MN is employed in different types of problems to evaluate its performance (in terms of speed and accuracy) against other popular neural fuzzy inference system. In the first category, performance comparison is carried out on a synthetic nonlinear dynamic system identification problem ([11], [20], [21]) followed by the second category where the performance is presented for benchmark time-series forecasting problems such as Mackey Glass [22], Box Jenkins problem [23] and monthly sunspot number prediction problem [23].

For performance comparison, Root Mean Square Error (RMSE) or Non-Destructive Error Index (NDEI) are used as performance metrics according to the standard practice.

A. Benchmark Dynamical System Identification Problems

First, a nonlinear dynamical system identification problem (SID) and its performance comparison results are presented.

Nonlinear System Identification I Problem: The SID I problem is a popular one taken from [20], [21]. The one step ahead output $y(k+1)$ depends on three past outputs and two past inputs. Problem details can be found in [21] the data generated for 900 time steps are used for training the network. Among them, the first 350 samples use a *iid* sequence uniform over $[-2, 2]$ as input and the remaining time steps use $sin\left(\frac{\pi k}{45}\right)$ as an input to the system. Using input-output realization [20], [12], [24] the number of inputs to the network are five, whereas in McRFIS-MN uses only two inputs to predict $y(k+1)$, i.e, $u(k)$ and $y(k)$ as memory neurons take care of the required past instances.

After the training, the network is tested with the following input signal,

$$u(k) = \begin{cases} \sin(\frac{\pi k}{25}) & \text{where} \quad 0 \leq k \leq 250 \\ +1.0 & \text{where} \quad 251 \leq k \leq 500 \\ -1.0 & \text{where} \quad 501 \leq k \leq 750 \\ 0.3\sin(\frac{\pi k}{25}) + 0.01\sin(\frac{\pi k}{32}) + 0.6\sin(\frac{\pi k}{10}) \end{cases} \tag{22}$$

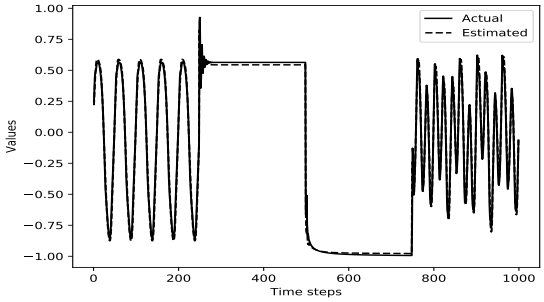

Figure 2: SID1 target vs actual plot with McRFIS-MN

Figure 2 shows the actual output of the discrete time dynamical system vs McRFIS-MNs predicted output. From the figure, one can easily observe that the McRFIS-MN follows the actual output very closely. The testing RMSE is 0.04. The memory neurons in the input and defuzzification layer helps McRFIS-MN to approximate the dynamics closely. McRFIS-MN is about ten times faster than its predecessors while maintaining comparable testing RMSE owing to its distinctive projection based learning through time, which numerically learns the optimal weights in one shot.

Table I: Performance comparison on SID-1 problem

Problem	Network	#Rules	Testing RMSE	CPU time(s)
SID1	eTS [25]	49	0.021	3
	SimpleTS [26]	22	0.030	5
	SAFIS [11]	17	0.022	4
	McFIS [12]	10	0.030	7
	McRFIS-MN	**14**	**0.040**	**0.46**

The performance of the McRFIS-MN is compared with existing results of type-1 fuzzy neural networks. The number of rules, testing RMSE are reported in Table I. From the table, one can see that McRFIS-MN achieves better performance than other state-of-the-art fuzzy neural networks primarily in terms of speed, while maintaining comparable test RMSE. Also, the number of rules in McRFIS-MN is smaller due to its self-adaptive structure.

B. Benchmark Time Series Problems

Next, performance comparison is performed on three benchmark chaotic time series forecasting problems, viz., the

Mackey-Glass [22], Box-Jenkins gas furnace problem [23] and monthly sunspot number prediction problem [27].

Mackey-Glass Time Series Prediction: The chaotic time series data is produced from the differential equation given in [22] as below,

$$\frac{dy}{dk} = \frac{0.2y(k-\bar{\tau})}{1+y^{10}(k-\bar{\tau})} - 0.1y(k) \tag{23}$$

The objective of the problem is to predict the 85 time steps ahead future value $y(k+85)$ based on past and current values, $y(k-18), y(k-12), y(k-6), y(k)$. The parameters are set as: $\bar{\tau} = 17$ and $y(0) = 1.2$. 3500 samples were produced out of which first 3000 were used for training and remaining 500 were used for testing the performance of McRFIS-MN. During training, instead of using all 4 past values as inputs only $y(k)$ is used. Please note, here instead of 1 step delay, 6 steps delayed outputs are used as inputs.

The performance of the prediction is measured using the non-destructive error index (NDEI is the ratio of the Test RMSE and the standard deviation of the test data; in this case, which is 0.51). Figure 3 shows the target vs actual plot from the McRFIS-MN netwok whereas table II provides the rule numbers, CPU time(s)and test-NDEI. From this table it can be observed that McRFIS-MN outperforms the other approaches in terms of both accuracy and runtime, while using a similar number of rules.

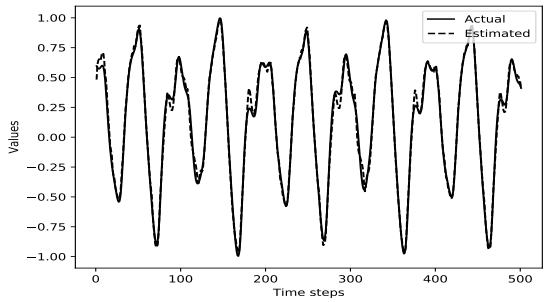

Figure 3: MG target vs actual plot with McRFIS-MN.

Box-Jenkins Gas Furnace Problem: The prediction of CO_2 emission based on input gas flow rate in the Box-Jenkins furnace can be cast as time series prediction problem. The prediction problem can be represented as in [23]. Note, $y(k)$ is the CO_2 concentration and $u(k)$ is the gas flow rate. Here, the input to McRFIS-MN is both $y(k)$ and $u(k)$. Memory neurons take care of the required lags. Based on the problem guidelines, the first 200 samples are used for training and the remaining 90 samples are used for testing. We see that the runtime is ten times faster while also achieving better Test RMSE.

Sunspot Time Series Prediction: Dark blotches on the surface of the sun are referred to as sunspots, they were first discovered in the 1700s after the invention of the telescope. Sunspots account for many solar activities (i.e. change in solar magnetism etc.) yet the proper causalities behind this phenomenon are still not known and that's why it has been used as a popular benchmark time series problem for a long time [23]. The monthly American sunspot data used in this experiment, contains 778 samples for the range of 1944 December to 2009 October.[1] The first 699 samples are used for learning while the remaining 79 samples are employed for testing. The goal is to forecast an one month or single step ahead value of the sunspot number using past two month's values. As can be seen in table II the Root Mean Square Error (RMSE) is used as the performance measure.

Table II: Performance comparison on benchmark problems

Problem	Network	#Rules	Test NDEI	CPU time(S)
MG	eTS [25]	9	0.380	0.3
	SAFIS [11]	6	0.376	0.5
	SimpleTS [26]	11	0.394	0.4
	McFIS [12]	10	0.100	0.9
	McRFIS-MN	**14**	**0.110**	**0.3**
BJ			**Test RMSE**	
	eTS [25]	9	0.049	0.4
	SAFIS [11]	5	0.071	0.6
	SimpleTS [26]	5	0.049	3
	McFIS [12]	12	0.036	0.2
	McRFIS-MN	**5**	**0.033**	**0.04**
Sunspot			**Test RMSE**	
	eTS [25]	23	0.047	3.5
	SAFIS [11]	21	0.100	4.4
	SimpleTS [26]	20	0.050	3.2
	McFIS [12]	12	0.060	4.2
	McRFIS-MN	**5**	**0.044**	**0.15**

The performance of McRFIS-MN is compared with other Type 1 neuro-fuzzy methods available in the literature and presented in table II. From the table, it can be observed that McRFIS-MN was able to attain better or similar accuracy while taking very less time to be trained. Hence, it is apparent that McRFIS-MN is faster in comparison with other approaches used in the literature while maintaining its good prediction accuracy.

CONCLUSION

In this paper, a recurrent type self-evolving neuro-fuzzy inference system and its learning algorithm have been presented. The use of MNs throughout the network at a cellular level helps in capturing the input-output dynamical relationship closely. The projection-based one-shot learning is very effective and fast. The performance is evaluated based on standard system identification and benchmark time series problems. After comparison with other neuro-fuzzy inference methods such as SAFIS, eTS, SimpleTS, and McFIS etc. the results clearly point to a better performance in terms of training speed while achieving a competitive (if not better) accuracy. The future direction of this work will be in the domain of true online sequential learning and incorporating type 2 fuzzy inference into McRFIS-MN for better handling of uncertainty in real world time series problems.

[1] https://www.ngdc.noaa.gov/stp/space-weather/solar-data/solar-indices/sunspot-numbers/american/lists

REFERENCES

[1] N. Haldrup, F. S. Nielsen, and M. Ø. Nielsen, "A vector autoregressive model for electricity prices subject to long memory and regime switching," *Energy Economics*, vol. 32, no. 5, pp. 1044–1058, 2010.

[2] I. Colak, S. Sagiroglu, and M. Yesilbudak, "Data mining and wind power prediction: A literature review," *Renewable Energy*, vol. 46, pp. 241–247, 2012.

[3] E. M. Azoff, *Neural network time series forecasting of financial markets*. John Wiley & Sons, Inc., 1994.

[4] H. Lütkepohl, *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[5] J. J. Hopfield, "Artificial neural networks," *IEEE Circuits and Devices Magazine*, vol. 4, no. 5, pp. 3–10, 1988.

[6] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[7] P. A. Mastorocostas and J. B. Theocharis, "A recurrent fuzzy-neural model for dynamic system identification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 32, no. 2, pp. 176–190, 2002.

[8] C.-F. Juang and C.-T. Lin, "A recurrent self-organizing neural fuzzy inference network," *IEEE Transactions on Neural Networks*, vol. 10, no. 4, pp. 828–845, 1999.

[9] P. Sastry, G. Santharam, and K. Unnikrishnan, "Memory neuron networks for identification and control of dynamical systems," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 306–319, 1994.

[10] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," in *Readings in Fuzzy Sets for Intelligent Systems*. Elsevier, 1993, pp. 387–403.

[11] H.-J. Rong, N. Sundararajan, G.-B. Huang, and P. Saratchandran, "Sequential adaptive fuzzy inference system (safis) for nonlinear system identification and prediction," *Fuzzy sets and systems*, vol. 157, no. 9, pp. 1260–1275, 2006.

[12] K. Subramanian and S. Suresh, "A meta-cognitive sequential learning algorithm for neuro-fuzzy inference system," *Applied soft computing*, vol. 12, no. 11, pp. 3603–3614, 2012.

[13] M. Pratama, S. G. Anavatti, P. P. Angelov, and E. Lughofer, "Panfis: A novel incremental learning machine," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 1, pp. 55–68, 2014.

[14] Y. Zhang and M. J. Er, "Sequential active learning using meta-cognitive extreme learning machine," *Neurocomputing*, vol. 173, pp. 835–844, 2016.

[15] M. Pratama, J. Lu, E. Lughofer, G. Zhang, and M. J. Er, "An incremental learning of concept drifts using evolving type-2 recurrent fuzzy neural networks," *IEEE Transactions on Fuzzy Systems*, vol. 25, no. 5, pp. 1175–1192, 2017.

[16] C. Zain, M. Pratama, E. Lughofer, and S. G. Anavatti, "Evolving type-2 web news mining," *Applied Soft Computing*, vol. 54, pp. 200–220, 2017.

[17] P. J. Werbos, "Backpropagation through time: what it does and how to do it," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.

[18] Y. Iiguni, H. Sakai, and H. Tokumaru, "A real-time learning algorithm for a multilayered neural network based on the extended kalman filter," *IEEE Transactions on Signal processing*, vol. 40, no. 4, pp. 959–966, 1992.

[19] G. S. Babu and S. Suresh, "Meta-cognitive rbf network and its projection based learning algorithm for classification problems," *Applied Soft Computing*, vol. 13, no. 1, pp. 654–666, 2013.

[20] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on neural networks*, vol. 1, no. 1, pp. 4–27, 1990.

[21] C.-F. Juang, "A tsk-type recurrent fuzzy network for dynamic systems processing by neural network and genetic algorithms," *IEEE Trans. on Fuzzy Systems*, vol. 10, no. 2, pp. 155–170, Apr 2002.

[22] R. S. Crowder, "Predicting the mackey-glass time series with cascade-correlation learning," in *Proc. 1990 Connectionist Models Summer School*. Carnegie Mellon Univ., Pittsburgh, PA, 1990, pp. 117–123.

[23] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[24] J. de Jesús Rubio, "Sofmls: online self-organizing fuzzy modified least-squares network," *IEEE Trans. on Fuzzy Systems*, vol. 17, no. 6, pp. 1296–1309, 2009.

[25] P. P. Angelov and D. P. Filev, "An approach to online identification of takagi-sugeno fuzzy models," *IEEE Trans. on Systems, Man, and Cybernetics, Part B*, vol. 34, no. 1, pp. 484–498, Feb 2004.

[26] P. Angelov and D. Filev, "Simpl_ets: a simplified method for learning evolving takagi-sugeno fuzzy models," in *Fuzzy Systems, 2005. FUZZ'05. The 14th IEEE International Conference on*. IEEE, 2005, pp. 1068–1073.

[27] Q. Song, "Robust initialization of a jordan network with recurrent constrained learning," *IEEE trans. on neural networks*, vol. 22, no. 12, pp. 2460–2473, 2011.