Self-Boosting First-Order Autonomous Learning Neuro-Fuzzy Systems

Xiaowei Gu [1,2] and Plamen Angelov [1,2,3*]

[1.] School of Computing and Communications, Lancaster University, Lancaster, LA1 4WA, UK

[2.] Lancaster Intelligent, Robotic and Autonomous Systems Centre (LIRA), Lancaster University, UK

[3.] Honorary Professor, Technical University, Sofia, 1000, Bulgaria

Email: {x.gu3, p.angelov}@lancaster.ac.uk

* Corresponding Author

Abstract: In this paper, a detailed mathematical analysis of the optimality of the premise and consequent parts of the recently introduced first-order Autonomous Learning Multi-Model (ALMMo) neuro-fuzzy system is conducted. A novel self-boosting algorithm for structure- and parameter- optimization is, then, introduced to the ALMMo, which results in the self-boosting ALMMo (SBALMMo) neuro-fuzzy system. By minimizing the objective functions with the previously collected data, the SBALMMo is able to optimize its system structure and parameters in few iterations. Numerical examples based benchmark datasets and real-world problems demonstrate the effectiveness and validity of the SBALMMo, and show the strong potential of the proposed approach for real applications.

Index Terms: neuro-fuzzy systems, autonomous learning, local optimality, self-boosting, streaming data processing.

1. Introduction

The initial concepts of evolving intelligent systems (EISs) were conceived around the turn of the 21st century [1]–[3] and now matured [4]. Nowadays, EISs have been widely applied for real-world problems [5]–[8].

EISs can be implemented in neuro-fuzzy [2], [9] or fuzzy rule-based structural forms [3]. The main difference between the EISs and the conventional (neuro-) fuzzy systems [10], [11] is their ability of self-organizing and self-updating system structure and parameters in real-time mode. Thus, EISs are usually associated with streaming data processing and can effectively approximate the dynamically changing environment [12]. Currently, the EISs is an intensively researched area [2], [3], [6], [13]–[18]. The most popular (neuro-) fuzzy systems include, but not limited to, the evolving Takagi-Sugeno systems (eTS) [1], [3], DENFIS [2], SAFIS [17], PANFIS [16], GENFIS [19] as well as the more recently introduced CENFS [18] and versions of the eTS (simpl_eTS [13], eTS+[20]). Interested readers are referred to the recent surveys [21], [22] for more details regarding the EISs.

Most of the EISs are of "one-pass" type, and they target processing streaming data "on the fly" [6], [13]–[15], [17], [20]. Although different EISs use different online learning mechanisms to evolve their premise (IF) parts, the majority of them use the recursive least squares (RLS) algorithm [23] or its extension, i.e. fuzzily weighted recursive least squares (FWRLS) algorithm [3] to learn the consequent parameters (THEN part).

Despite that the concepts of the EISs have been matured [4] and many of them have been applied in real-world scenarios successfully, a systematic mathematical study on the local optimality of the premise (IF) and consequent (THEN) parts of the current EISs has not been conducted yet [24]. The main reason for this is obvious, as EISs are implemented for handling the data streams with ever-changing data pattern in complex, nonstationary environments [12]. Currently, only the stability of the consequent (THEN) part of EISs has been proven [3], [25].

Another problem that the majority of the EISs suffer from is the predefined problem- and user-specific parameters [2], [5], [9], [16]–[18], [26] for the structure learning algorithms. These parameters require *prior* knowledge and/or the assumptions about the nature of the problems to be defined in advance. In addition, they also influence the objectiveness of the learning results.

First-order Autonomous Learning Multi-Model (ALMMo) neuro-fuzzy system [6] is a new type of multi-model system introduced on the basis of the first-order AnYa type neuro-fuzzy systems [4], [27]. The ALMMo system is composed of fully human intelligible IF-THEN rules of AnYa type, and its structure is built upon the nonparametric data clouds that are free of external constraints enabling the ALMMo system to objectively approximate the real data distribution. Its system structure and meta-parameters are derived from the data directly without imposing data generation model with user- or problem- specific parameters. Thus, there is no requirement of *prior* knowledge of the problem. The system is also memory- and computation- efficient. Its system structure is able to evolve online, and its meta-parameters can be recursively updated. Meanwhile, ALMMo is capable of monitoring the quality of the dynamically evolving structure in real time, which guarantees the computation- and memory-efficiency of the learning mechanism. All of these allow the ALMMo neuro-fuzzy system to effectively and efficiently follow the possible shifts and/or drifts in the data pattern for the case of streaming data [6].

Based on a systematic analysis of the local optimality of the premise (IF) and consequent (THEN) parts of the ALMMo neuro-fuzzy system, in this paper, we introduce a novel self-boosting algorithm to the ALMMo, and we call the upgraded ALMMo neuro-fuzzy system: self-boosting Autonomous Learning Multi-Model (SBALMMo). The concept of a “self-boosting” system means that the system is able to self-optimize its structure and meta-parameters, and, thus, achieves a better performance. The self-boosting algorithm involves two independent iterative processes for minimizing the objective functions designed for the premise (IF) and consequent (THEN) parts based on the historical observations. The proposed SBALMMo is able to guarantee the local optimality. Moreover, we also introduce a novel quality monitoring mechanism to the SBALMMo, which assists the system to discard stale fuzzy rules, and thus, further strengths the ability of the SBALMMo to follow the shifts and/or drifts of the data pattern [12]. To summarize, the main advantages of the SBALMMo compared with the ALMMo neuro-fuzzy system include:

1) It guarantees periodic local optimization for both the premise (IF) and consequent (THEN) parts of the neuro-fuzzy system on a solid mathematical basis.

2) It introduces a dynamical quality monitoring mechanism to the rule base.

The remainder of this paper is organized as follows. Section 2 briefly recalls the general architecture and principles of the ALMMo. The local optimality analysis of the ALMMo neuro-fuzzy system is conducted in section 3 with the optimization approaches for the both parts of the system given in the same section. The SBALMMo is described in section 4 and the numerical examples are presented in section 5. Section 6 concludes the paper.

2. The ALMMo Neuro-Fuzzy System

In this section, the general architecture and learning process of the ALMMo neuro-fuzzy system are briefly recalled to make this paper self-contained. In this paper, we use the Euclidean distance by default if there is no special declaration.

2.1. General Architecture

The general architecture of the ALMMo neuro-fuzzy system is depicted in Fig. 1 [6], [27].

$$IF\ (\boldsymbol{x}\sim\boldsymbol{p})\quad THEN\ (y=\bar{\boldsymbol{x}}^T\boldsymbol{a}) \tag{1}$$

where $\boldsymbol{x}=[x_1,x_2,\dots,x_M]^T\in\mathbf{R}^M$ is the input vector of the fuzzy rule; $\mathbf{R}^M$ is the M dimensional real data space; $\sim$ denotes the similarity, or a fuzzy degree of satisfaction/membership [27]; $\bar{\boldsymbol{x}}^T=[1,\boldsymbol{x}^T]$; $\boldsymbol{p}=[p_1,p_2,\dots,p_M]^T$ is the prototype of this fuzzy rule; $\boldsymbol{a}=[a_0,a_1,\dots,a_M]^T$ is the vector of consequent parameters.

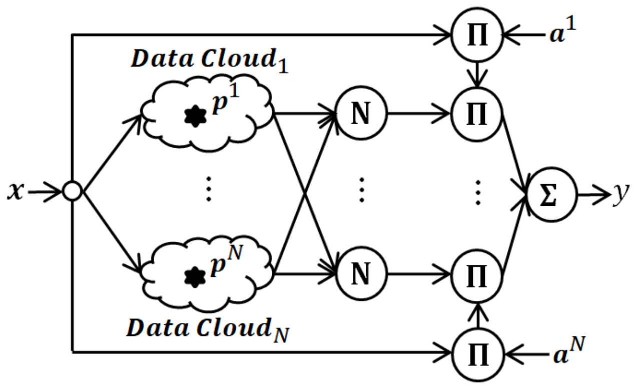


Fig. 1. General Architecture of ALMMo fuzzy system

The ALMMo fuzzy system is mathematically modelled as [3], [6], [27]:

$$y_K = f(\boldsymbol{x}_K) = \sum_{i=1}^{N_K} \lambda_K^i \overline{\boldsymbol{x}}_K^T \boldsymbol{a}_K^i \tag{2}$$

where y_K and $\boldsymbol{x}_K$ ($\boldsymbol{x}_K \in \mathbf{R}^M$) are the respective output and input vectors of the ALMMo at the *K*-th time instance (we assume that only one data sample arrives at each time instance); $\overline{\boldsymbol{x}}_K^T = [1, \boldsymbol{x}_K^T\ ]$; N_K denotes the number of fuzzy rules/data clouds; $\boldsymbol{a}_K^i = \left[a_{K,0}^i, a_{K,1}^i, \dots, a_{K,M}^i\right]^T$ is the consequent parameter vector of the *i*-th fuzzy rule at the *K*-th time instance; λ_K^i is the corresponding firing strength derived from the normalized data density directly from the data [3], [6], [27], [28].

2.2. The Learning Process Summary

In this subsection, the main procedure of the learning process of the ALMMo neuro-fuzzy system, which includes both, the system identification and the algorithmic parameters identification processes, is summarized as follows. One can find the more detailed algorithm description in the original paper [6].

Step 0. System initialization

The ALMMo neuro-fuzzy system is initialized by the first data sample, $\boldsymbol{x}_1$ ($K \leftarrow 1$).

The global meta-parameters of the system, namely, the global mean, $\boldsymbol{\mu}_1$, global average scalar product, X_1 and the number of existing fuzzy rules/data clouds, N_1 are initialized as:

$$\boldsymbol{\mu}_1 \leftarrow \boldsymbol{x}_1; \quad X_1 \leftarrow \|\boldsymbol{x}_1\|^2; \quad N_1 \leftarrow 1 \tag{3}$$

The first data cloud, $\mathbf{C}_1^1$ is initialized by $\boldsymbol{x}_1$: $\mathbf{C}_1^1 \leftarrow \{\boldsymbol{x}_1\}$. The meta-parameters of $\mathbf{C}_1^1$, which include the prototype, $\boldsymbol{p}_1^1$, average scalar product, X_1^1, support, S_1^1, accumulated firing strength, Λ_1^1, utility, η_1^1 and the time instance at which $\mathbf{C}_1^1$ is initialized, I^1 are initialized as:

$$\boldsymbol{p}_1^1 \leftarrow \boldsymbol{x}_1; \quad X_1^1 \leftarrow \|\boldsymbol{x}_1\|^2; \quad S_1^1 \leftarrow 1; \quad \Lambda_1^1 \leftarrow 1; \quad \eta_1^1 \leftarrow 1; \quad I^1 \leftarrow 1 \tag{4}$$

and the consequent parameters, $\boldsymbol{a}_1^1$ and covariance matrix, $\boldsymbol{\Theta}_1^1$ of the corresponding fuzzy rule of $\mathbf{C}_1^1$, $\boldsymbol{R}^1$ are initialized as:

$$\boldsymbol{a}_1^1 \leftarrow \mathbf{0}_{(M+1)\times 1}; \quad \boldsymbol{\Theta}_1^1 \leftarrow \Omega_o \mathbf{I}_{(M+1)\times(M+1)} \tag{5}$$

The first fuzzy rule $\boldsymbol{R}^1$ within the fuzzy rule-base is, then, initialized as:

$$\boldsymbol{R}^1: \quad IF\ (\boldsymbol{x} \sim \boldsymbol{p}_1^1) \quad THEN\ (y = \overline{\boldsymbol{x}}^T \boldsymbol{a}_1^1) \tag{6}$$

Step 1. System output generation

When a new data sample, $\boldsymbol{x}_K$ is available ($K \leftarrow K+1$), the local data density of $\boldsymbol{x}_K$ within each data cloud ($j = 1,2,\dots,N_{K-1}$) is calculated as [6]:

$$D_{K-1}^j(\boldsymbol{x}_K) = \frac{1}{1 + \frac{\left(S_{K-1}^j\right)^2 \left\|x_K - p_{K-1}^j\right\|^2}{\left(S_{K-1}^j + 1\right)\left(S_{K-1}^j X_{K-1}^j + \|x_K\|^2\right) - \left\|x_K + S_{K-1}^j p_{K-1}^j\right\|^2}} \tag{7}$$

Local data density calculated by equation (7) reacts to the potential changes in the data pattern quickly by incorporating $\boldsymbol{x}_K$ into the existing data clouds and shifting the corresponding prototypes towards it. This enlarges the role $\boldsymbol{x}_K$ plays in the system output and improves the system's adaptive ability.

Then, the firing strength of each fuzzy rule to $\boldsymbol{x}_K$ is calculated ($j = 1,2,\dots,N_{K-1}$):

$$\lambda_{K-1}^{j} = \frac{D_{K-1}^{j}(\boldsymbol{x}_K)}{\sum_{k=1}^{N_{K-1}} D_{K-1}^{k}(\boldsymbol{x}_K)} \tag{8}$$

and the system output is calculated and exported by equation (9):

$$y_K = \sum_{i=1}^{N_K} \lambda_{K-1}^{i} \bar{\boldsymbol{x}}_K^T \boldsymbol{a}_{K-1}^{i} \tag{9}$$

Step 2. Global meta-parameter updating

The global mean, $\boldsymbol{\mu}_{K-1}$ and the average scalar product, X_{K-1} of system are, then, updated:

$$\boldsymbol{\mu}_K = \frac{K-1}{K}\boldsymbol{\mu}_{K-1} + \frac{1}{K}\boldsymbol{x}_K \tag{10}$$

$$X_K = \frac{K-1}{K}X_{K-1} + \frac{1}{K}\|\boldsymbol{x}_K\|^2 \tag{11}$$

The data densities at $\boldsymbol{x}_K$ and $\boldsymbol{p}_{K,-1}^{j}$ ($j = 1,2,\dots,N_{K-1}$) are calculated by equation (12a),

$$D_K(\boldsymbol{z}) = \frac{1}{1+\frac{\|\boldsymbol{z}-\boldsymbol{\mu}_K\|^2}{X_K-\|\boldsymbol{\mu}_K\|^2}}; \quad \boldsymbol{z} = \boldsymbol{x}_K, \boldsymbol{p}_{K-1}^{1}, \boldsymbol{p}_{K-1}^{2}, \dots, \boldsymbol{p}_{K-1}^{N_{K-1}} \tag{12a}$$

As one can see, the data density calculated by equation (12a) has a more elegant form compared with the local data density calculated by equation (7). This is because the global mean, $\boldsymbol{\mu}_K$ and the average scalar product, X_K have both been updated with $\boldsymbol{x}_K$ by equations (10) and (11) before the calculation. By substituting $\boldsymbol{\mu}_K$ and X_K with $\boldsymbol{\mu}_{K-1}$ and X_{K-1}, respectively, equation (12a) can be formulated in a similar form as equation (7):

$$D_K(\boldsymbol{z}) = \frac{1}{1+\frac{(K-1)^2\|\boldsymbol{z}-\boldsymbol{\mu}_{K-1}\|^2}{K\left((K-1)X_{K-1}+\|\boldsymbol{z}\|^2\right)-\|\boldsymbol{z}+(K-1)\boldsymbol{\mu}_{K-1}\|^2}}; \quad \boldsymbol{z} = \boldsymbol{x}_K, \boldsymbol{p}_{K-1}^{1}, \boldsymbol{p}_{K-1}^{2}, \dots, \boldsymbol{p}_{K-1}^{N_{K-1}} \tag{12b}$$

Step 3. System structure updating

Firstly, **Condition 1** is checked to see whether $\boldsymbol{x}_K$ is the prototype of a new data cloud [3]:

Condition 1:
$$\begin{gathered} IF\left(D_K(\boldsymbol{x}_K) > \max_{j=1,2,\dots,N_{K-1}} D_K\left(\boldsymbol{p}_{K-1}^{j}\right)\right) \\ OR\left(D_K(\boldsymbol{x}_K) < \min_{j=1,2,\dots,N_{K-1}} D_K\left(\boldsymbol{p}_{K-1}^{j}\right)\right) \\ THEN\ (\boldsymbol{x}_K\ is\ a\ new\ prototype) \end{gathered} \tag{13}$$

The rationale behind **Condition 1** is that if $D_K(\boldsymbol{x}_K)$ is larger than the data densities at the prototypes of the existing data clouds, $\boldsymbol{x}_K$ is more descriptive and it has more summarization power than all the other prototypes. In contrast, if $D_K(\boldsymbol{x}_K)$ is smaller than the minimum data density at any of the prototypes, $\boldsymbol{x}_K$ represents a new pattern that no existing data clouds can describe. Therefore, in both cases, $\boldsymbol{x}_K$ becomes a new prototype and a new data cloud is initialized by it.

If **Condition 1** is met, **Condition 2** is, then, checked to see whether this new data cloud is overlapping with any of the existing data clouds:

Condition 2: $$IF\left(D_{K-1}^{j}(\boldsymbol{x}_K) \geq 0.8\right) \quad THEN\left(\boldsymbol{p}_{K-1}^{j}\ is\ very\ close\ to\ \boldsymbol{x}_K\right) \tag{14}$$

If both, **Conditions 1 and 2** are satisfied, the nearest data cloud, $\mathbf{C}_{K-1}^{n*}$ to $\boldsymbol{x}_K$ is identified by equation (15):

$$n* = \operatorname{argmin}_{j=1,2,..,N_{K-1}}\left(\left\|\boldsymbol{x}_K - \boldsymbol{p}_{K-1}^{j}\right\|\right) \tag{15}$$

where $\left\|\boldsymbol{x}_K - \boldsymbol{p}_{K-1}^{j}\right\| = \sqrt{\sum_{l=1}^{M}\left(x_{K,l} - \boldsymbol{p}_{K-1,l}^{j}\right)}$ denotes the Euclidean distance between $\boldsymbol{x}_K$ and $\boldsymbol{p}_{K-1}^{j}$.

$\boldsymbol{x}_K$ is assigned to $\mathbf{C}_{K-1}^{n*}$ ($\mathbf{C}_K^{n*} \leftarrow \mathbf{C}_{K-1}^{n*} + \boldsymbol{x}_K$) with the meta-parameters of $\mathbf{C}_{K-1}^{n*}$, including the prototype, $\boldsymbol{p}_{K-1}^{n*}$, average scalar product, X_{K-1}^{n*} and support, S_{K-1}^{n*}, updated as:

$$\boldsymbol{p}_K^{n*} \leftarrow \frac{\boldsymbol{p}_{K-1}^{n*}+\boldsymbol{x}_K}{2}; \quad X_K^{n*} \leftarrow \frac{X_{K-1}^{n*}+\|\boldsymbol{x}_K\|^2}{2}; \quad S_K^{n*} \leftarrow \left\lceil\frac{S_{K-1}^{n*}+1}{2}\right\rceil \tag{16}$$

If **Condition 1** is met, **Condition 2** is unsatisfied, a new data cloud with $\boldsymbol{x}_K$ as the prototype is added to the system ($N_K \leftarrow N_{K-1}+1$): $\mathbf{C}_K^{N_K} \leftarrow \{\boldsymbol{x}_K\}$, and the meta-parameters of $\mathbf{C}_K^{N_K}$ are initialized as:

$$\boldsymbol{p}_K^{N_K} \leftarrow \boldsymbol{x}_1; \quad X_K^{N_K} \leftarrow \|\boldsymbol{x}_1\|^2; \quad S_K^{N_K} \leftarrow 1; \quad \Lambda_{K-1}^{N_K} \leftarrow 0; \quad \eta_{K-1}^{N_K} \leftarrow 1; \quad I^{N_K} \leftarrow K \tag{17}$$

The consequent parameters of the fuzzy rule corresponding to $\mathbf{C}_K^{N_K}$ are initialized as:

$$\boldsymbol{a}_{K-1}^{N_K} \leftarrow \frac{1}{N_{K-1}} \sum_{\boldsymbol{j=1}}^{N_{K-1}} \boldsymbol{a}_{K-1}^j; \quad \boldsymbol{\Theta}_{K-1}^{N_K} \leftarrow \Omega_o \mathbf{I}_{(M+1)\times(M+1)} \tag{18}$$

where Ω_o is a user-controlled parameter initializing covariance matrix [29]. This is representing the standard for recursive least squares algorithms. The recommended setting for Ω_o is given at the end of this section.

The fuzzy rule corresponding to $\mathbf{C}_K^{N_K}$ is initialized as:

$$\boldsymbol{R}^{N_K}: \quad IF\ \left(\boldsymbol{x} \sim \boldsymbol{p}_{K-1}^{N_K}\right) \quad THEN\ \left(y = \overline{\boldsymbol{x}}^T \boldsymbol{a}_{K-1}^{N_K}\right) \tag{19}$$

Otherwise, when, both, **Conditions 1 and 2** are unsatisfied, the nearest data cloud, $\mathbf{C}_{K-1}^{n*}$ to $\boldsymbol{x}_K$ is identified by equation (15), and the meta-parameters are updated as:

$$S_K^{n*} \leftarrow S_{K-1}^{n*}+1; \quad \boldsymbol{p}_K^{n*} \leftarrow \frac{S_{K-1}^{n*}}{S_K^{n*}} \boldsymbol{p}_{K-1}^{n*} + \frac{1}{S_K^{n*}} \boldsymbol{x}_K; \quad X_K^{n*} \leftarrow \frac{S_{K-1}^{n*}}{S_K^{n*}} X_{K-1}^{n*} + \frac{1}{S_K^{n*}} \|\boldsymbol{x}_K\|^2 \tag{20}$$

For each $\mathbf{C}_{K-1}^j \in \{\mathbf{C}\}$ that does not receive new member at this time instance, there is $\mathbf{C}_K^j \leftarrow \mathbf{C}_{K-1}^j$, and its meta-parameters are set as:

$$\boldsymbol{p}_K^j \leftarrow \boldsymbol{p}_{K-1}^j; \quad X_K^j \leftarrow X_{K-1}^j; \quad S_K^j \leftarrow S_{K-1}^j \tag{21}$$

Step 4. Fuzzy rule quality monitoring

The local data density of $\boldsymbol{x}_K$ within each data cloud is calculated by the following equation ($j = 1,2,\ldots,N_K$):

$$D_K^j(\boldsymbol{x}_K) = \frac{1}{1+\frac{\left\|x_K - p_K^j\right\|^2}{X_K^j - \left\|p_K^j\right\|^2}} \tag{22}$$

and the firing strength of each fuzzy rule is calculated ($j = 1,2,\ldots,N_K$):

$$\lambda_K^j = \frac{D_K^j(x_K)}{\sum_{k=1}^{N_K} D_K^k(x_K)} \tag{23}$$

Then, the accumulated firing strength Λ_{K-1}^j is updated to Λ_K^j for each fuzzy rule ($j = 1,2,\ldots,N_K$):

$$\Lambda_K^j \leftarrow \Lambda_{K-1}^j + \lambda_K^j \tag{24}$$

and the utility η_K^j of each fuzzy rule is calculated ($j = 1,2,\ldots,N_K$):

$$\eta_K^j \leftarrow \frac{1}{K-I^j} \Lambda_K^j \tag{25}$$

After this, **Condition 3** is, then, checked to remove the data cloud and fuzzy rule that contributes little to the overall system output [4], [20]:

$$\textbf{Condition 3:}\ IF\ \left(\eta_K^j < \eta_o\right) \quad THEN\ \left(\mathbf{C}_K^j\ and\ \boldsymbol{R}^j\ are\ removed\right) \tag{26}$$

where η_o is another user-controlled parameter for monitoring the quality of fuzzy rules. The recommended setting of η_o is given at the end of this section.

If $\mathbf{C}_K^j$ and $\boldsymbol{R}^j$ satisfy **Condition 3**, they are removed from the system and fuzzy rule base with $N_K \leftarrow N_K - 1$.

Step 5. Consequent parameter updating

The consequent parameters of the fuzzy rules in the rule-base are updated using the FWRLS method as ($j = 1,2,\ldots,N_K$) [3]:

$$\boldsymbol{\Theta}_K^j \leftarrow \boldsymbol{\Theta}_{K-1}^j - \frac{\lambda_K^j \boldsymbol{\Theta}_{K-1}^j \overline{\boldsymbol{x}}_K \overline{\boldsymbol{x}}_K^T \boldsymbol{\Theta}_{K-1}^j}{1+\lambda_K^j \overline{\boldsymbol{x}}_K^T \boldsymbol{\Theta}_{K-1}^j \overline{\boldsymbol{x}}_K} \tag{27}$$

$$\boldsymbol{a}_K^j \leftarrow \boldsymbol{a}_{K-1}^j + \lambda_K^j \boldsymbol{\Theta}_{K-1}^j \overline{\boldsymbol{x}}_K \left(y_K - \overline{\boldsymbol{x}}_K^T \boldsymbol{a}_{K-1}^j\right) \tag{28}$$

Step 6. Fuzzy rule-base updating

Update the fuzzy rules in the rule base with the newly updated antecedent and consequent parameters ($j = 1,2,\ldots,N_K$):

$$\boldsymbol{R}^j: \quad IF\ (\boldsymbol{x} \sim \boldsymbol{p}_K^j) \quad THEN\ (y = \overline{\boldsymbol{x}}^T \boldsymbol{a}_K^j) \tag{29}$$

After this, the system goes back to **Step 1** if new data samples are available.

As it was analysed in [6], Ω_o is used for initializing the covariance matrix for each new fuzzy rule added to the rule base, and η_o is used for monitoring the quality of the existing fuzzy rules.

In practice, η_o subtly influences the system structure. The recommended value range of η_o is $[0, 0.1]$ [4]. The larger η_o is, the faster the system removes the fuzzy rules that contribute very little to the overall system output from its rule base, the more efficient the system will be, and vice versa. However, it may deteriorate the performance as the system forgets the acquired knowledge too fast if the value of η_o is too large [4]. While Ω_o influences the convergence of the consequent part. The system performance may deteriorate if the value of Ω_o is set too large or too small [3].

Same as [6], in this paper, we use $\Omega_o = 10$ and $\eta_o = 0.1$. However, we have to stress that both of them have marginal influence on the performance of ALMMo neuro-fuzzy system as well as on the optimization process, which will be introduced in the later sections. This will be empirically verified through numerical examples in section 5.1.

3. Local Optimality Analysis of the ALMMo

It has been proven in [25] that the stability of (neuro-)fuzzy system is always guaranteed on condition that the FWRLS algorithm is used for consequent parameters updating globally. Furthermore, this proof can also be applied to the FWRLS algorithm for consequent parameters updating per rule [25]. However, there is no guarantee on the optimality of the parameters of such kind of NFSs (in both the premise (IF) and consequent (THEN) parts) within a finite number of training samples in a nonstationary environment. Optimality is of great importance for a learning system, but is seldom analysed because of the complex nature of the problem itself.

In this section, we will discuss the optimality of the ALMMo system. However, it has to be stressed that the optimality analysis also applies to other fuzzy systems of AnYa type that use FWRLS algorithm for consequent parameter updating. As the ALMMo comprises of the premise (IF) part and the consequent (THEN) part that are identified via two independent processes, the optimality of the two parts are analysed separately.

3.1. Optimization of the Premise Part

The optimality of the premise (IF) part of a traditional (Mamdani type [10] or Takagi-Sugeno type [30]) fuzzy system is not an easy question to answer because of the handcrafting involved in the design process of these fuzzy systems, i.e. the linguistic terms and the membership functions. However, the important simplification of the AnYa type fuzzy rule by replacing the traditional premise part with a simpler form (a prototype) [27] provides the possibility to systematically study the system optimality. This is because the optimality of the premise (IF) part of an AnYa type neuro-fuzzy system is reduced to the problem of obtaining the optimal partition of the data space, which can be considered as achieving locally optimal clustering. The formal mathematical condition for this can be formulated in the form of a mathematical programming problem [31].

Considering that the ALMMo neuro-fuzzy system partitions $\{\boldsymbol{x}\}_K$ into N_K data clouds, the local optimization problem of the ALMMo is formulated as the following mathematical programming problem for clustering/data partitioning [31]:

$$\textbf{Problem 1}: F(\boldsymbol{W}_K, \boldsymbol{P}_K) = \sum_{i=1}^{N_K} \sum_{j=1}^{K} w_K^{i,j} \left\| \boldsymbol{x}_j - \boldsymbol{p}_K^i \right\|, \tag{30}$$

where $\boldsymbol{W}_K = [w_K^{i,j}]$ is a $N_K \times K$ real matrix; $\boldsymbol{P}_K = [\boldsymbol{p}_K^1, \boldsymbol{p}_K^2, \dots, \boldsymbol{p}_K^{N_K}] \in \mathbf{R}^{M \times N_K}$ is a $M \times N_K$ matrix consisting of all the identified prototypes at the K-th time instance. $\boldsymbol{W}_K$ is subject to the following constrains ($i = 1,2,\dots,N_K, j = 1,2,\dots,K$):

$$w_K^{i,j} \geq 0\ and\ \sum_{i=1}^{N_K} w_K^{i,j} = 1. \tag{31}$$

Problem 1 is a nonconvex problem and the local minimum point does not need to be global minimum [31]. A local minimum point of $F(\boldsymbol{W}_K, \boldsymbol{P}_K)$ is a locally optimal solution of **Problem 1**.

It can be mathematically proven that the prototype-based (neuro-)fuzzy systems of "one-pass" type do not guarantee the locally optimal solutions after the learning process, which is also applies to the ALMMo. At the *i*-th time instance, the data sample, $\boldsymbol{x}_i$ $(i < K)$ is observed, there is a guarantee that $\boldsymbol{x}_i$ is assigned to the nearest prototype, $\boldsymbol{p}_i^{n_i}$ thanks to equation (15) [32]:

$$\sum_{j=1}^{N_i} w_i^{j,i} \| \boldsymbol{x}_i - \boldsymbol{p}_i^j \| = \| \boldsymbol{x}_i - \boldsymbol{p}_i^{n_i} \| \tag{32}$$

where n_i denote the index of the nearest prototype to $\boldsymbol{x}_i$ at the *i*-th time instance; $\mathbf{C}_i^{n_i}$ is the corresponding data cloud which $\boldsymbol{x}_i$ is assigned to, namely $\boldsymbol{x}_i \in \mathbf{C}_i^{n_i}$.

However, with the new data samples arrive, $\boldsymbol{x}_{i+1}, \boldsymbol{x}_{i+2}, \dots, \boldsymbol{x}_K$, the prototype, $\boldsymbol{p}_i^{n_i}$, would not stay the same because of the shifts and/or drifts of the data pattern [12], which means that:

$$\boldsymbol{p}_i^{n_i} \neq \boldsymbol{p}_K^{n_i} \ and \ \| \boldsymbol{x}_i - \boldsymbol{p}_i^{n_i} \| \neq \| \boldsymbol{x}_i - \boldsymbol{p}_K^{n_i} \| \tag{33}$$

In addition, because of the newly observed data samples, new data clouds will also be initialized, and the following inequality is established:

$$\| \boldsymbol{x}_i - \boldsymbol{p}_K^{n_i} \| \geq \| \boldsymbol{x}_i - \boldsymbol{p}_K^{n*} \| = \min_{j=1,2,\dots,N_K} \left(\| \boldsymbol{x}_i - \boldsymbol{p}_K^j \| \right) \tag{34}$$

However, because of the "one-pass" type nature of the learning process of the ALMMo neuro-fuzzy system, once a data sample is assigned to a particular data cloud, it is discarded by the system and a later re-partition is not possible. Considering this and combining equations (32) and (34), there is:

$$\sum_{j=1}^{N_K} w_K^{j,i} \| \boldsymbol{x}_i - \boldsymbol{p}_K^j \| = \| \boldsymbol{x}_i - \boldsymbol{p}_K^{n_i} \| \geq \min_{j=1,2,\dots,N_K} \left(\| \boldsymbol{x}_i - \boldsymbol{p}_K^j \| \right) \tag{35}$$

Therefore, one can conclude that $\sum_{j=1}^{N_K} w_K^{j,i} \| \boldsymbol{x}_i - \boldsymbol{p}_K^j \|$ does not achieve the minimum value at the *K*-th time instance and the partition is not a locally optimal solution.

To solve this problem, we propose a feasible solution for ALMMo system to achieve local optimality by further applying an iterative process to minimize $F(\boldsymbol{W}_K, \boldsymbol{P}_K)$, i.e. using the well-known K-means clustering algorithm [31], [32]. To make this paper self-contained, we summarize the iterative processing in the following pseudo code.

Algorithm 1

```
While $F(\widehat{\boldsymbol{W}}_K, \widehat{\boldsymbol{P}}_K) \neq F(\boldsymbol{W}_K, \boldsymbol{P}_K)$
    i. Obtain $\boldsymbol{W}_K$ based on $\boldsymbol{P}_K$ by the following expression ($i = 1,2,\dots,N_K, j = 1,2,\dots,K$):
        $$\begin{cases} w_K^{i,j} = 1 & i = \operatorname{argmin}_{l=1,2,\dots,N_K} \left( \| \boldsymbol{x}_j - \boldsymbol{p}_K^l \|^2 \right) \\ w_K^{i,j} = 0 & i \in else \end{cases} \tag{36}$$
    ii. Calculate $F(\boldsymbol{W}_K, \boldsymbol{P}_K)$ using equation (30);
    iii. Obtain $\widehat{\boldsymbol{P}}_K$ based on $\boldsymbol{W}_K$ by the following expression ($i = 1,2,\dots,N_K$):
        $$\widehat{\boldsymbol{p}}_K^i = \frac{1}{\sum_{j=1}^{K} w_K^{i,j}} \sum_{j=1}^{K} w_K^{i,j} \boldsymbol{x}_j \tag{37}$$
    iv. Obtain $\widehat{\boldsymbol{W}}_K$ based on $\widehat{\boldsymbol{P}}_K$ by equation (36);
    v. Calculate $F(\widehat{\boldsymbol{W}}_K, \widehat{\boldsymbol{P}}_K)$ using equation (30);
    vi. Set $\boldsymbol{P}_K \leftarrow \widehat{\boldsymbol{P}}_K$;
End While
```

By applying **Algorithm 1** to the partitioning results obtained by the ALMMo neuro-fuzzy system, one can always guarantee the local optimality of the premise (IF) part. Furthermore, numerical examples in section 5 demonstrates that **Algorithm 1** can converge to the locally optimal solution in a finite number of iterations (the detailed theoretical proof can be found in [31]).

3.2. Optimization of the Consequent Part

The stability of the (neuro-)fuzzy system with consequent parameters updated globally using the FWRLS algorithm has been analysed in [25]. It has been proven that the error between the system output and the reference output, which is caused by the parameter errors, converges to a small neighbourhood of zero in which the average identification error satisfies [25]:

$$\lim_{K\to\infty}\frac{1}{K}\sum_{k=1}^{K}e_k^2 \le \varepsilon_0 \tag{38}$$

where $e_k = \hat{y}_k - y_k$; $\hat{y}_k$ and y_k are the system output and reference output; ε_0 is a certain upper bound of the average identification error. The detailed proof can be found in [25].

The consequent parameters of the ALMMo neuro-fuzzy system are updated using the FWRLS algorithm per sub-model (fuzzy rule). The locally updated FWRLS (per fuzzy rule) is significantly less influenced by the system structure evolution than the globally updated FWRLS and is significantly less computationally expensive [3], [4]. Therefore, the locally updated FWRLS is more likely to achieve the theoretical optimality for the RLS condition [3], [4].

In practice, with a finite number of observations ($K \ll \infty$), a "one-pass" type parameter learning mechanism is usually unable to guarantee an optimal solution. This is because of the lack of an iterative search process, which allows the consequent parameters converging to the locally optimal. Therefore, a common practice is to approximate the locally optimal parameters by repeating the learning process until the overall training error converges into an acceptable range [33].

At the K-th time instance ($K \ll \infty$), the overall error function of the learning process is formulated as the following mathematical programming problem [34]:

$$\textbf{Problem 2}: E(\hat{\boldsymbol{y}},\boldsymbol{y}) = \frac{1}{K}\sum_{k=1}^{K}e_k^2 \tag{39}$$

With the identified N_K fuzzy rules with the respective prototypes denoted as $\boldsymbol{p}_K^i$ ($i = 1,2,\ldots,N_K$), **Problem 2** can be reformulated as:

$$E(\hat{\boldsymbol{y}},\boldsymbol{y}) = \frac{1}{K}\sum_{k=1}^{K}\left(\sum_{i=1}^{N_K}\lambda_k^i\bar{\boldsymbol{x}}_k^T\boldsymbol{a}_k^i - y_k\right)^2 \tag{40}$$

where λ_k^i is calculated using equation (23) ($k = 1,2,\ldots,K$) and is fixed throughout the whole iteration process since the prototypes ($\boldsymbol{p}_K^i$) stay the same.

To have a convergence of the overall error function $E(\hat{\boldsymbol{y}},\boldsymbol{y})$ into a small range close to 0 with the observations $\{y\}_K$ and $\{\boldsymbol{x}\}_K$, we use the following iterative process by repeating the consequent parameters updating process, which is essentially equivalent to approximate $\boldsymbol{a}_k^i$ ($k = 1,2,\ldots,K,\ i = 1,2,\ldots,N_K$) to the optimal values throughout the whole iteration, see **Algorithm 2** below (ε_1 is the stopping threshold for the iterative process, which is a small number close to 0, in this paper, we use $\varepsilon_1 = 10^{-5}$).

Algorithm 2

```
i. Set $\boldsymbol{a}_0^i = \left[\overbrace{0,0,\ldots,0}^{M+1}\right]$ and $\boldsymbol{\Theta}_0^i = \Omega_o\mathbf{I}_{(M+1)\times(M+1)}$ ($i = 1,2,\ldots,N_K$);
ii. Calculate $\lambda_k^i$ ($i = 1,2,\ldots,N_K$, $k = 1,2,\ldots,K$) using equation (23);
iii. While $\|E_1 - E_0\| \ge \varepsilon_1$:
        * Set $k \leftarrow 1$;   $E_0 \leftarrow E_1$;   $E_1 \leftarrow 0$;
        * While $k \le K$:
            - Read $y_k$ and $\boldsymbol{x}_k$;
            - $E_1 \leftarrow E_1 + \left(\sum_{i=1}^{N_K}\lambda_k^i\bar{\boldsymbol{x}}_k^T\boldsymbol{a}_{k-1}^i - y_k\right)^2$;
            - Update $\boldsymbol{a}_{k-1}^i$ and $\boldsymbol{\Theta}_{k-1}^i$ to $\boldsymbol{a}_k^i$ and $\boldsymbol{\Theta}_k^i$ using FWRLS algorithm (equations (27) and (28));
            - $k \leftarrow k + 1$;
        * End While
        * Set $E_1 \leftarrow \frac{1}{K} E_1$;
        * Set $\boldsymbol{a}_0^i \leftarrow \boldsymbol{a}_K^i$ and $\boldsymbol{\Theta}_0^i \leftarrow \boldsymbol{\Theta}_K^i$;
    iv. End While
```

We have to admit that in practice, it is very hard to let the overall error function $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ converge to 0 due to the complexity of the real problems. In section 5 we will demonstrate that using **Algorithm 2**, the overall error function will converge to a small nonzero value in a few iterations.

4. The Proposed SBALMMo Neuro-Fuzzy System

In this section, we propose a novel self-boosting algorithm to upgrade the ALMMo neuro-fuzzy system, and thus, we introduce the SBALMMo. A new type of quality monitoring mechanism is also added. It has to be stressed that the self-boosting and quality monitoring mechanisms are actually very flexible and universal, and they can be applied to other neuro-fuzzy and fuzzy rule-based systems of AnYa type with the FWRLS consequent parameters updating algorithm. On the other hand, we have to admit that with the self-boosting algorithm, the learning process of the SBALMMo becomes incremental instead of being of "one-pass" type.

4.1. Self-Boosting Algorithm with Quality Monitoring Mechanism

In this subsection, we will describe the self-boosting algorithm for the SBALMMo in detail, which is conducted on a chunk-by-chunk basis. Nonetheless, alternative approaches to activate the self-boosting can be considered as well, i.e. the self-boosting can be activated at each time the structure and parameters of SBALMMo are self-updated (which will be computationally expensive) or when the error between the system output and the reference output exceeds a certain threshold [24]. Moreover, there are different ways for the system to use the observed data, i.e. the learning system can choose to keep all the previously collected data chunks in memory for self-boosting, and, thus, optimize the system with all the previously gained information; the learning system may also discard the processed chunks and only use the most recently collected chunk (or a few recent ones) to follow the time-changing data pattern effectively.

During the learning process of the ALMMo neuro-fuzzy system presented in section 2, the processed data samples are discarded automatically to relief the memory burden. However, with the self-boosting algorithm, the ALMMo sends each processed data sample and the corresponding reference output sequentially to a data pool with the maximum capacity of Q (which can be in external memory). It has to be stressed that Q is not a problem- or user-specific parameter.

Once the maximum capacity of the data pool is reached, which means the data pool has collected the most recently arrived Q data samples and the corresponding reference outputs, the ALMMo will encapsulate the Q data samples and outputs into a data chunk, denoted by $\{\acute{\boldsymbol{x}}, \acute{y}\}_Q$ and empty the pool for the next round.

Before the ALMMo processes the next new data sample (between **Steps 5 and 6**), the ALMMo boosts itself with the data chunk $\{\acute{\boldsymbol{x}}, \acute{y}\}_Q$ by applying **Algorithm 1** firstly to optimize the positions of the identified prototypes in the data space, namely, obtain the locally optimal prototypes. After this, the premise (IF) part of the ALMMo is optimized based on $\{\acute{\boldsymbol{x}}, \acute{y}\}_Q$, and the optimized prototypes are re-denoted by $\acute{\boldsymbol{p}}_K^1, \acute{\boldsymbol{p}}_K^2, \ldots, \acute{\boldsymbol{p}}_K^{N_K}$.

Then, we need to identify the stale fuzzy rules that are less activated before **Algorithm 2** is used for optimizing the consequent parameters further. The stale fuzzy rules are usually generated due to the shifts and/or drifts of the changing data pattern [12]. We calculate the activation level of each data sample within $\{\acute{\boldsymbol{x}}, \acute{y}\}_Q$ to each fuzzy rule using equation (23), denoted by $\acute{\lambda}_k^i$ $(i = 1,2, \ldots, N_K, k = 1,2, \ldots, Q)$ with the optimized prototypes. The stale fuzzy rules are identified using **Condition 4:**

Condition 4: $IF\ \left(\sum_{k=1}^{Q} \acute{\lambda}_k^i < (1 - \Omega_1) \cdot \acute{\lambda}_0\right)\ \ THEN\ \left(\acute{\boldsymbol{p}}_K^i\ is\ a\ stale\ prototype\right)$ (41)

where $i = 1,2, \ldots, N_K$; $\acute{\lambda}_0 = \frac{1}{N_K} \sum_{k=1}^{Q} \sum_{i=1}^{N_K} \acute{\lambda}_k^i$; Ω_1 is a small value ($|\Omega_1| < 1$), and the recommended value range of Ω_1 is $0 \leq \Omega_1 \leq 0.1$, which means that the SBALMMo neuro-fuzzy system will discard up to half of the identified prototypes and the corresponding fuzzy rules that are less activated.

Condition 4 is used for identifying the fuzzy rules that are activated much less compared with others, which indicates that the prototypes of these fuzzy rules are not as representative as others, and they do not stand for the local modes of the data density anymore. Once a prototype, for instance, $\acute{\boldsymbol{p}}_K^i$, is recognized as a stale one, the corresponding data cloud will be removed from the system structure, and the fuzzy rule will be removed from the rule base as well.

In contrast with other criteria for identifying stale fuzzy rules, for example, the utilities [4], [20] (**Condition 3** in this paper) or the ages [12] of the fuzzy rules, **Condition 4** is able to react quickly to the shifts and/or drifts [12] of the data pattern because it concerns only the more recently observed data samples instead of all the previously observed data set.

Following the quality monitoring, **Algorithm 2** is applied to optimize the consequent parameters further. After both, the premise (IF) and consequent (THEN) parts of the ALMMo have been optimized, the system goes back to the normal status and processes the newly arrived data samples on a sample-by-sample basis until the data pool is full again.

4.2. Algorithm Summary

The main procedure of the self-boosting algorithm used in the SBALMMo system is summarized as follows.

Self-Boosting Algorithm

```
i. Send the processed input and output to the data pool;
ii. If (data pool is full) Then:
    * Encapsulate $\{\acute{\boldsymbol{x}}, \acute{y}\}_Q$ and empty the data pool;
    * Execute Algorithm 1;
    * Remove stale fuzzy rules using Condition 4;
    * Execute Algorithm 2;
iii. End If
```

The flowchart of the learning process of the SBALMMo neuro-fuzzy system is presented in Fig. 2, where the self-boosting algorithm is executed in **Step 5.5**.

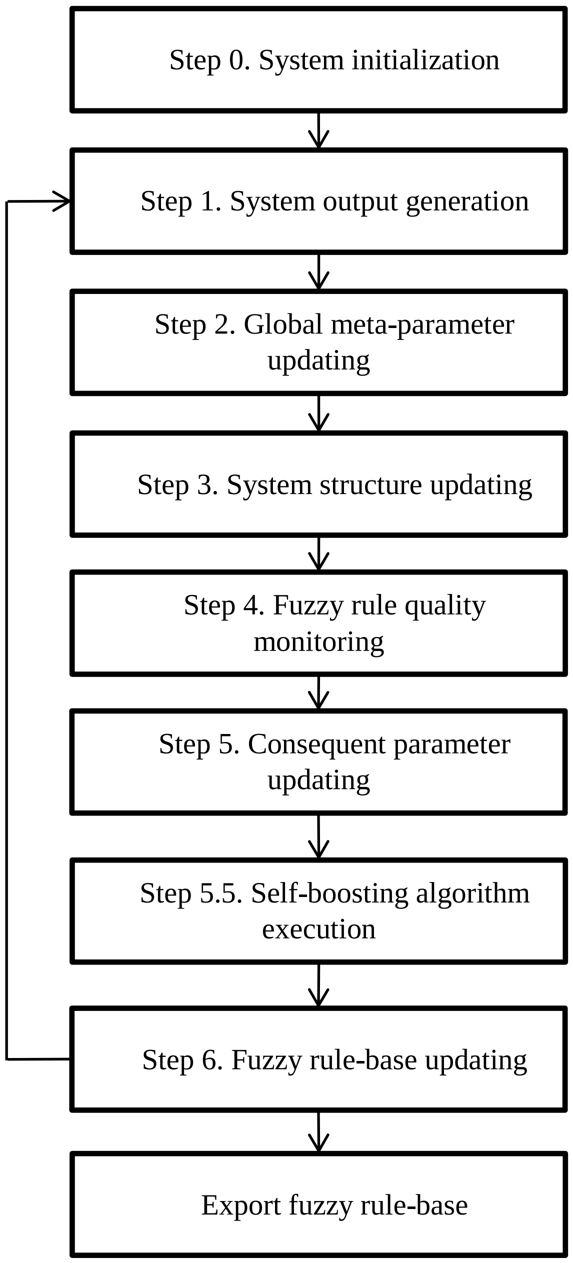

Step 0. System initialization
Step 1. System output generation
Step 2. Global meta-parameter updating
Step 3. System structure updating
Step 4. Fuzzy rule quality monitoring
Step 5. Consequent parameter updating
Step 5.5. Self-boosting algorithm execution
Step 6. Fuzzy rule-base updating
Export fuzzy rule-base

Fig. 2. The flowchart of the learning process of the SBALMMo neuro-fuzzy system

4.3. Computational Complexity Analysis

In this subsection, the computational complexity of each step of the SBALMMo neuro-fuzzy system (as presented in Fig.2) is analysed. However, since the system structure of the proposed approach is dynamically changing all the time, we assume that the computational complexity analysis is conducted at the K-th time instance.

Step 0: The computational complexity of this step is negligible since this step concerns the system initialization only, and the computation related to this step would not be performed if $K > 1$.

Step 1: The computational complexity of calculating the local data density-based firing strength is $O(N_{K-1}W)$ thanks to the recursive calculation form (equation (7)), and the complexity of producing the system output is $O(N_{K-1}W)$ as well. Therefore, the computational complexity of this step is $O(N_{K-1}W)$.

Step 2: The computational complexities for updating the global meta-parameters ($\boldsymbol{\mu}_K$ and X_K) and calculating the data densities are $O(N_{K-1}W)$ and $O((N_{K-1}+1)W)$, respectively, and thus, the computational complexity of this step is $O((N_{K-1}+1)W)$.

Step 3: This step mainly concerns updating the existing fuzzy rule or adding a rule to the rule base, and thus, the computational complexity of this step is $O(W)$.

Step 4: The most of the computations in this step are related to calculating the firing strength of each fuzzy rule in the rule base. Therefore, similarly to the steps 1 and 2, the computational complexity of this step is $O(N_K W)$.

Step 5: Updating the consequent parameters of fuzzy rules requires significantly more computational resources because the FWRLS method requires the covariance matrices, $\boldsymbol{\Theta}_K^j (j = 1,2,\dots,N_K)$ to be updated, and the computational complexity of this step is $O(N_K W^2)$.

Step 5.5: This is the extra step that the proposed SBALMMo system has in comparison with the ALMMo system. **Algorithms 1 and 2** in this step consume most of the computational resources though they are activated only when the data pool is full.

When the data pool reaches its storage capacity, **Algorithm 1** is used firstly to optimize the premise parts of the fuzzy rules with the accumulated Q historical data samples based on the distances between the historical data samples and the prototypes of the fuzzy rules. Therefore, the computational complexity of **Algorithm 1** is $O(QN_K W)$.

After **Algorithm 1** has been finished and the stale fuzzy rules have been removed**, Algorithm 2** is executed. **Algorithm 2** is for optimizing the consequent parts of the fuzzy rules, and the most of the computation resources are used for updating the covariance matrices for the fuzzy rules. Therefore, the computational complexity of **Algorithm 2** is $O(QN_K W^2)$.

Therefore, the overall computational complexity of step 5.5 is $O(QN_K W^2)$ when **Algorithms 1 and 2** are activated. Otherwise, the computation of step 5.5 is negligible.

Step 6: This step updates fuzzy rules in the rule base with the latest premise and consequent parameters, and the computational complexity of this step is negligible as well.

5. Numerical Examples & Discussions

In this section, a number of numerical examples are performed to validate the proposed concepts and the method. The overall performance of the proposed approach is evaluated on benchmark datasets as well as on real-world problems. Details of the comparative algorithms used in this paper are provided in Table 1.

The algorithms were developed using MATLAB R2018a, the performance was evaluated on a PC with dual core processor 3.60 GHz×2 and 16 GB RAM.

5.1. Performance on Benchmark Regression Problems

In this subsection, firstly, the following widely used benchmark datasets are used for numerical examples to demonstrate the performance of the proposed SBALMMo:

1) Autos;

2) Triazines;

3) Delta Ailerons, and

4) California Housing.

The details of the problems are given in Table 2. Following the common practice, the input and output attributes are normalized to the range of [0,1] in advance [18], [25].

Table 1. Details of the comparative algorithms

Abbreviation	Full Name	Online/ Offline Training	Parameter Setting
AnYa	AnYa neuro-fuzzy system with eClustering learning algorithm [27]	Online	Same as in [27]
CEFNS	Correntropy-based evolving fuzzy neural system[18]	Online	Same as in [18]
DENFIS	Dynamic evolving neural-fuzzy inference system [2]	Online	Same as in [2]
ESAFIS	Extended sequential adaptive fuzzy inference system [35]	Online	Same as in [35]
eTS	Evolving Takagi-Sugeno fuzzy model [3]	Online	Same as in [3]
FCMMS	Fuzzily connected multi-model system [36]	Online	Same as in [36]
McFIS	Meta-cognitive neuro-fuzzy inference system [37]	Online	Same as in [37]
PANFIS	Parsimonious network based on fuzzy inference system[16]	Online	Same as in [16]
SAFIS	Sequential adaptive fuzzy inference system [17]	Online	Same as in [17]
SeroFAM	Self-reorganizing fuzzy-associative learning system [38]	Online	Same as in [38]
Simpl_eTS	A simplified method for learning evolving Takagi-Sugeno fuzzy model [13]	Online	Same as in [13]
EFuNN	Evolving fuzzy neural network [39]	Offline	Same as in [39]
OLSLR	Least square linear regression algorithm [40]		
SWLSLR	Sliding window least square linear regression algorithm [41]		Window size: 200 samples

Table 2. Details of the benchmark datasets for regression

Dataset	Attributes	Training Samples, K_T	Validation Samples, K_V
Autos	15 inputs + 1 output	80	79
Triazines	60 inputs + 1 output	100	86
Delta Ailerons	5 inputs + 1 output	3000	4129
California Housing	8 inputs + 1 output	10320	10320

In the first numerical example, the relationship between the size of the data pool, Q and the performance of the SBALMMo is investigated. In this example, the SBALMMo is trained based on the training set and, then, tested on the validation set. During the validation stage, the SBALMMo stops updating itself. Since the cardinality of the four benchmark problems are different, to make the results comparable, we set $Q = \frac{K_T}{5}, \frac{K_T}{4}, \frac{K_T}{3}, \frac{K_T}{2}, K_T$ and $\Omega_1 = 0$, and report the results in terms of root mean square error (RMSE), number of fuzzy rules (rule#) and execution time (t_{exe}, in seconds) in Table 3. The average time consumption for each self-boosting during the training stage is given to illustrate the computational complexity of the proposed approach. We also report the results obtained by ALMMo as a baseline.

Table 3. Performance on benchmark regression datasets using different value of Q ($\Omega_1 = 0$)

Algorithm		SBALMMo					ALMMo
		Q					
Dataset	Measure	$\frac{K_T}{5}$	$\frac{K_T}{4}$	$\frac{K_T}{3}$	$\frac{K_T}{2}$	K_T	
Autos	RMSE	0.0879	0.0509	0.0494	0.0457	0.0452	0.0568
	rule#	2	3	3	4	4	8
	t_{exe} (s)	0.30	0.32	0.38	0.30	0.18	0.02
	t_{exe} per Self-boosting (s)	0.05	0.08	0.12	0.14	0.17	

Triazines	RMSE	0.0411	0.0322	0.0070	0.0046	0.0022	0.0078
	rule#	2	2	1	1	3	7
	t_{exe} (s)	2.18	1.18	1.41	1.40	2.48	0.19
	t_{exe} per Self-boosting (s)	0.40	0.28	0.39	0.64	2.34	
Delta Ailerons	RMSE	0.0513	0.0513	0.0513	0.0511	0.0512	0.0512
	rule#	1	1	2	5	4	10
	t_{exe} (s)	1.30	1.71	1.44	1.79	1.78	0.47
	t_{exe} per Self-boosting (s)	0.17	0.30	0.32	0.65	1.08	
California Housing	RMSE	0.0773	0.0773	0.0771	0.0771	0.0771	0.0777
	rule#	5	5	2	2	5	10
	t_{exe} (s)	9.56	10.49	7.31	5.47	6.96	1.84
	t_{exe} per Self-boosting (s)	1.62	2.21	1.95	1.92	5.46	

Remark 1: Table 3 demonstrates that, in general, the SBALMMo with the data pool of a larger size is able to achieve a better regression performance in terms of RMSE. This is because a smaller data pool can only provide a small proportion of data samples for optimization process, and the structure and meta-parameters of the SBALMMo are more likely to converge to a local minimum solution that is only optimal for this small proportion of data only. By using a larger data pool, the proposed SBALMMo can achieve a better performance since the premise (IF) and consequent (THEN) parts of the ALMMo system are optimized based on the majority of the data (even the whole dataset). On the other hand, a larger data pool will require more time for the self-boosting algorithm to reach the locally optimal solution. This is in accordance with the computational complexity analysis in subsection 4.3. A smaller data pool allows the SBALMMo system to be able to perform the self-boosting algorithm more efficiently, but, since the self-boosting algorithm has to be activated more frequently, this might increase the overall computational time.

Secondly, the relationship between the forgetting factor, Ω_1 and the performance of the SBALMMo is investigated based on the four benchmark problems. Similar to the previous example, in this example, the SBALMMo is trained based on the training set and, then, tested on the validation set. We set $Q = K_T$ and change the value of Ω_1 as $\Omega_1 = -0.02, 0, 0.02, 0.04, 0.06, 0.08$ and 0.1. The root mean square error (RMSE) and number of fuzzy rules (rule#) of the regression results are reported in Table 4.

Table 4. Performance on benchmark regression datasets using different value of Ω_1 ($Q = K_T$)

Algorithm		**SBALMMo**							**ALMMo**
Dataset	**Measure**	Ω_1							
		-0.02	0	0.02	0.04	0.06	0.08	0.10	
Autos	RMSE	0.0455	0.0452	0.0452	0.0452	0.0452	0.0446	0.0446	0.0568
	rule#	3	4	5	5	5	6	6	8
Triazines	RMSE	0.0022	0.0022	0.0022	0.0022	0.0023	0.0023	0.0022	0.0078
	rule#	3	3	3	3	5	6	7	7
Delta Ailerons	RMSE	0.0512	0.0512	0.0512	0.0512	0.0512	0.0512	0.0512	0.0512
	rule#	2	4	9	10	10	10	10	10
California Housing	RMSE	0.0771	0.0771	0.0770	0.0770	0.0770	0.0770	0.0770	0.0777
	rule#	1	5	9	9	10	10	10	10

Remark 2: Table 4 demonstrates that the performance of the SBALMMo is less influenced by the value of Ω_1, generally. However, with a smaller value of Ω_1, the system complexity of the SBALMMo is largely decreased.

Based on Remarks 1 and 2, we use $= K_T$,$\Omega_1 = 0$ as the default setting in the following numerical examples of this subsection,

In the following two numerical examples, the influence of the user-controlled parameters, Ω_0 and η_o on the performance of the SBALMMo system is investigated. Following the same experimental protocol, firstly, we use the recommended value of η_o, namely, $\eta_o = 0.1$, and change the value of Ω_0 as follows: $\Omega_0 = 1,5,10,25$ and 100. The root mean square error (RMSE) and number of fuzzy rules (rule#) of the regression results are reported in Table 5.

Table 5. Performance on benchmark regression datasets using different value of Ω_0 ($\eta_o = 0.1$)

Dataset	**Measure**	Ω_0				
		1	5	10	25	100
Autos	RMSE	0.0447	0.0450	0.0452	0.0456	0.0460
	rule#	4	4	4	4	4
Triazines	RMSE	0.0032	0.0024	0.0022	0.0018	0.0013
	rule#	3	3	3	3	3
Delta Ailerons	RMSE	0.0512	0.0512	0.0512	0.0512	0.0512
	rule#	4	4	4	4	4
California Housing	RMSE	0.0772	0.0771	0.0771	0.0770	0.0770
	rule#	5	5	5	5	5

Then, we use the recommended value of Ω_0, namely, $\Omega_0 = 10$, and change the value of η_o as follows: $\eta_o = 0.01, 0.03, 0.05, 0.10$ and 0.20 to repeat the numerical experiments. The root mean square error (RMSE) and number of fuzzy rules (rule#) of the regression results are reported in Table 6.

Table 6. Performance on benchmark regression datasets using different value of η_o ($\Omega_0 = 10$)

Dataset	**Measure**	η_o				
		0.01	0.03	0.05	0.10	0.20
Autos	RMSE	0.0452	0.0452	0.0452	0.0452	0.0455
	rule#	4	4	4	4	3
Triazines	RMSE	0.0022	0.0022	0.0022	0.0022	0.0020
	rule#	3	3	3	3	3
Delta Ailerons	RMSE	0.0512	0.0512	0.0512	0.0512	0.0512
	rule#	15	15	11	4	2
California Housing	RMSE	0.0771	0.0771	0.0770	0.0771	0.0770
	rule#	3	3	3	5	3

Remark 3: Tables 5 and 6 demonstrate that, Ω_o has negligible influence on the optimization process. While, η_o has subtle influence on the optimization process since it changes the number of fuzzy rules in the rule base. Nonetheless, the performance of the SBALMMo system in terms of RMSE is only marginally influenced by both of them.

For a better demonstration of the effectiveness of the system structure and parameter optimization algorithm, the corresponding changes of the values of $F(\boldsymbol{W}_K, \boldsymbol{P}_K)$ and $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ during the self-boosting process on Autos, Triazines, Delta Ailerons and California Housing dataset are depicted in Fig. 3, where one can see that the values of $F(\boldsymbol{W}_K, \boldsymbol{P}_K)$ and $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ converge to the minimum in few iterations.

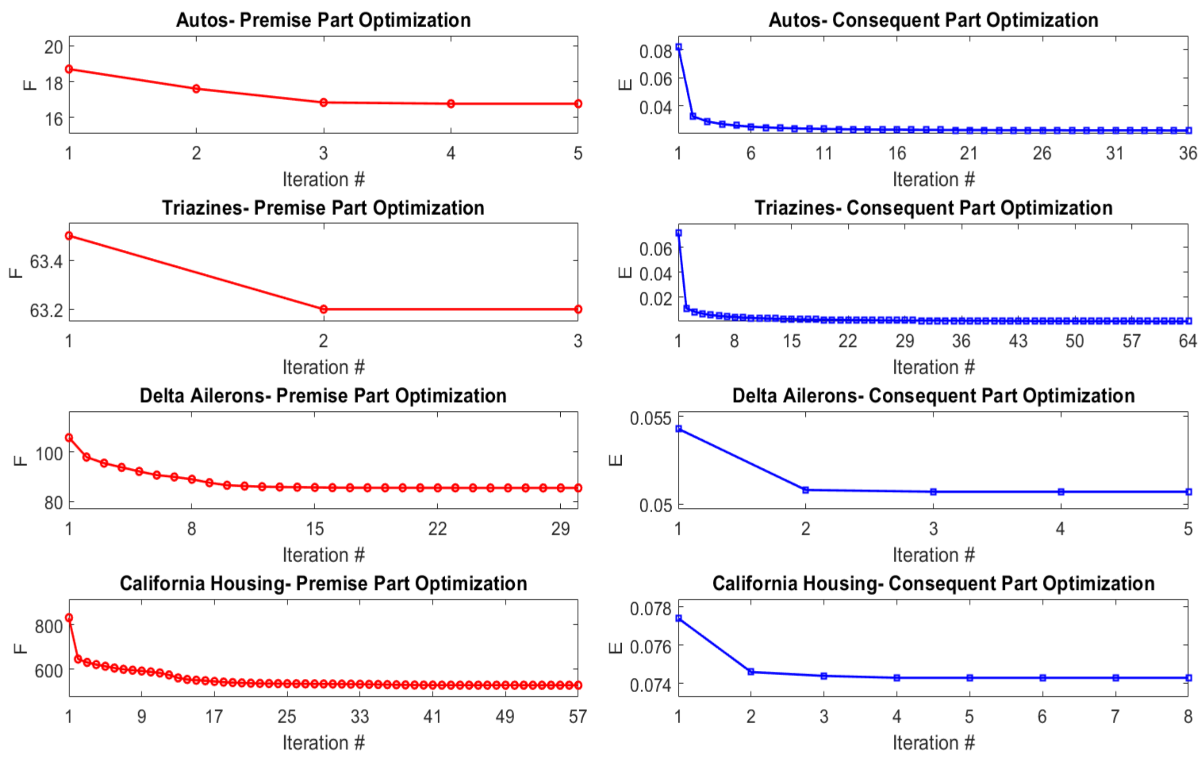


Fig. 3. Changes of the values of $f(\boldsymbol{W}_K, \boldsymbol{P}_K)$ and $E(\hat{\boldsymbol{y}}, \boldsymbol{y})$ during self-boosting operations.

The performance of the SBALMMo is further compared with the well-known "state-of-the-art" algorithms based on the four benchmark datasets, the results are reported in Table 7 in terms of root mean square error (RMSE) and number of fuzzy rules (rule#). All the algorithms involved in the comparison are trained with the training set and tested on the validation set. During the validation stage, the involved algorithms will stop updating. The parameter settings of these algorithms have been given in Table 1.

Table 7. Performance comparison on benchmark datasets

Dataset	Algorithm	RMSE	rule#	Dataset	Algorithm	RMSE	rule#
Autos	**SBALMMo**	**0.0452**	**4**	Delta Ailerons	**SBALMMo**	**0.0512**	**4**
	ALMMo [6]	0.0568	8		ALMMo [6]	0.0512	10
	AnYa [27]	0.0942	3		AnYa [27]	0.0983	2
	CEFNS [18]	0.0666	2		CEFNS [18]	0.0502	3
	ESAFIS [35]	0.0604	3		ESAFIS [35]	0.0506	13
	eTS [3]	0.0535	3		eTS [3]	0.0513	4
	McFIS [37]	0.0687	3		McFIS [37]	0.0509	15
	SAFIS [17]	0.1184	5		SAFIS [17]	0.0549	14
	Simpl_eTS [13]	0.0689	10		Simpl_eTS [13]	0.0512	4
Triazines	**SBALMMo**	**0.0022**	**3**	California Housing	**SBALMMo**	**0.0771**	**5**
	ALMMo [6]	0.0078	7		ALMMo [6]	0.0777	10
	AnYa [27]	0.0754	5		AnYa [27]	0.1044	4
	CEFNS [18]	0.0452	6		CEFNS [18]	0.0878	2
	ESAFIS [35]	0.0331	19		ESAFIS [35]	0.0892	6
	eTS [3]	0.0179	9		eTS [3]	0.0772	3
	McFIS [37]	0.0556	12		McFIS [37]	0.0822	15
	SAFIS [17]	0.0581	9		SAFIS [17]	0.0988	12
	Simpl_eTS [13]	0.0197	9		Simpl_eTS [13]	0.0773	3

As one can see from Table 7, the proposed SBALMMo neuro-fuzzy system is able to achieve the best performance on the Autos, Trazines and California Housing datasets. In addition, the SBALMMo neuro-fuzzy system has a lighter system structure compared with the ALMMo, which means that it is able to provide a better interpretability for users.

Furthermore, for a better evaluation, we randomly descramble the order of the training samples of the four benchmark datasets to train the SBALMMo neuro-fuzzy system and test the statistical performance based on the validation set. We also involve other well-known algorithms for comparison. The statistical results are given in Table 8 in terms of the means and standard deviations of the three measures, namely, RMSE, rule# and t_{exe}.

Table 8. Statistical result comparison on benchmark datasets

Dataset	**Algorithm**	**Mean ± Stand Deviation**		
		RMSE	**rule#**	**t_{exe} (s)**
Autos	**SBALMMo**	**0.0456±0.0002**	**3.6±0.9**	**0.16±0.05**
	ALMMo [6]	0.0549±0.0028	7.3±2.2	0.02±0.01
	FCMMS [36]	0.1238±0.0806	1.2±0.4	0.02±0.01
	AnYa [27]	0.1095±0.0305	3.0±0.5	0.01±0.01
	ESAFIS [35]	0.0437±0.0032	2.2±0.4	0.10±0.05
	SAFIS [17]	0.1752±0.0443	2.4+0.8	0.02±0.01
Triazines	**SBALMMo**	**0.0022±0.0001**	**3.5±0.8**	**2.12±0.71**
	ALMMo [6]	0.0092±0.0012	7.1±1.3	0.06±0.02
	FCMMS [36]	0.0326±0.0086	1.0±0.0	0.03±0.01
	AnYa [27]	0.0944±0.0304	5.5±0.5	0.14±0.06
	ESAFIS [35]	0.0103±0.0034	2.4±0.5	0.29±0.12
	SAFIS [17]	0.2033±0.0452	2.5±0.9	0.03±0.01
Delta Ailerons	**SBALMMo**	**0.0512±0.0000**	**5.0±1.0**	**1.51±0.34**
	ALMMo [6]	0.0516±0.0008	9.6±1.3	0.42±0.01
	FCMMS [36]	0.0703±0.0167	4.5±3.3	0.39±0.09
	AnYa [27]	0.1160±0.0274	2.0±0.0	0.11±0.02
	ESAFIS [35]	0.0508±0.0002	2.0±0.0	0.67±0.12
	SAFIS [17]	0.0652±0.0197	5.7±1.3	0.21±0.03
California Housing	**SBALMMo**	**0.0771±0.0000**	**3.8±1.4**	**5.23±1.54**
	ALMMo [6]	0.0782±0.0012	9.8±1.1	1.53±0.08
	FCMMS [36]	0.1013±0.0174	3.1±2.6	1.64±0.23
	AnYa [27]	0.1112±0.0278	3.7±0.5	0.70±0.11
	ESAFIS [35]	0.0789±0.0121	2.1±0.8	2.40±0.72
	SAFIS [17]	0.0939±0.0028	10.9±1.7	1.63±0.30

As one can see from the comparison, despite that the training samples are re-ordered randomly, the SBALMMo neuro-fuzzy system always performs better and more stable in terms of the values of the mean and the standard deviation of RMSE. This is because that the structure and the meta-parameters of the SBALMMo have, both, been optimized with the training data, and thus, the influence of the order of the training samples on the system is minimized.

Comparing between ALMMo and SBALMMo, one may notice that SBALMMo requires 3 to 35 times more execution time for the training process because of the iterative search process for the optimal solution. Nonetheless, it is noticeable that the average RMSE of the results obtained by the SBALMMo reduces by 1% to 75%, and the standard deviation of the RMSE is very close to zero. In addition, the number of fuzzy rules in the SBALMMo is only half of the number of rules generated by ALMMo. This means that the SBALMMo system is able to provide accurate regression results with a simpler system structure and much higher robustness. The simpler system structure further improves the human interpretability of the learning results.

5.2. Performance on Foreign Currency Exchange Data

In this subsection, we test the proposed SBALMMo neuro-fuzzy system on the following three of the most widely currency pairs trading all around world:

1) Euro versus United States Dollar (EUR/USD)

2) Australian Dollar versus United States Dollar (AUD/USD)

3) Euro versus United States Dollar (EUR/USD)

The historical currency exchange prices of the three currency pairs from the 2 January 2002 – 30 December 2016 are obtained from [42]. Each dataset (one per currency pair) is composed of the following five attributes:

1) data/time tag, K,

2) open price, $x_{K,1}$,

3) high price, $x_{K,2}$,

4) low price, $x_{K,3}$ and

5) close price, $x_{K,4}$.

In the following numerical example, the datasets are divided into a training set and a testing set with a ratio of $K_T : K_V = 7:3$. The open price, $x_{K,1}$, high price, $x_{K,2}$, low price, $x_{K,3}$ and close price, $x_{K,4}$ of the current day is used to predict the close price of the next data, namely, $\hat{x}_{K+1,4} = f\left(\boldsymbol{x}_K = \left[x_{K,1}, x_{K,2}, x_{K,3}, x_{K,4}\right]^T\right)$. The SBALMMo neuro-fuzzy system will stop updating its system structure and meta-parameters during the validation stage. In this subsection, Q and Ω_1 are set as $Q = K_T$,$\Omega_1 = 0$.

The prediction performance of the SBALMMo system on the currency exchange data is tabulated in Table 9 in terms of root mean square error (RMSE), number of fuzzy rules (rule#) and execution time (t_{exe}, in seconds). It is also compared with other well-known approaches under the same experimental protocol, and the comparison results are also reported in Table 9. We also report the results obtained by dummy prediction method in Table 9, which takes the closing price of the current day as the predicted closing price of the next day (namely, $\hat{x}_{K+1,4} \leftarrow x_{K,4}$), as the baseline. The prediction results obtained by SBALMMo are depicted in Fig. 4 for illustration.

Table 9. Performance comparison on foreign currency exchange data

	Algorithm	**RMSE**	**rule#**	**t_{exe} (s)**
EUR/USD	**SBALMMo**	**0.0060**	**4**	**1.03**
	ALMMo [6]	0.0062	8	0.46
	FCMMS [36]	0.0131	5	0.95
	AnYa [27]	0.0060	2	0.28
	eTS [3]	0.0065	6	31.98
	ESAFIS [35]	0.0066	2	1.28
	SAFIS [17]	0.0190	2	0.33
	Dummy [a]	0.0060		
AUD/USD	**SBALMMo**	**0.0050**	**5**	**1.52**
	ALMMo [6]	0.0052	7	0.42
	FCMMS [36]	0.0337	12	1.12
	AnYa [27]	0.0112	2	0.27
	eTS [3]	0.0050	8	32.80
	ESAFIS [35]	0.0050	2	1.39
	SAFIS [17]	0.0198	2	0.31
	Dummy	0.0050		
GBP/USD	**SBALMMo**	**0.0078**	**5**	**1.08**
	ALMMo [6]	0.0080	7	0.62

	FCMMS [36]	0.0110	9	1.23
	AnYa [27]	0.0085	2	0.31
	eTS [3]	0.0131	8	32.08
	ESAFIS [35]	0.0082	2	1.37
	SAFIS [17]	0.0427	2	0.29
	Dummy	0.0078		

[a] The prediction result obtained by the dummy method.

As one can see from Table 9, the SBALMMo is able to provide highly accurate prediction on all three currency exchange pairs outperforming other alternative approaches, which demonstrates the strong potential of the SBALMMo neuro-fuzzy system in real-world applications.

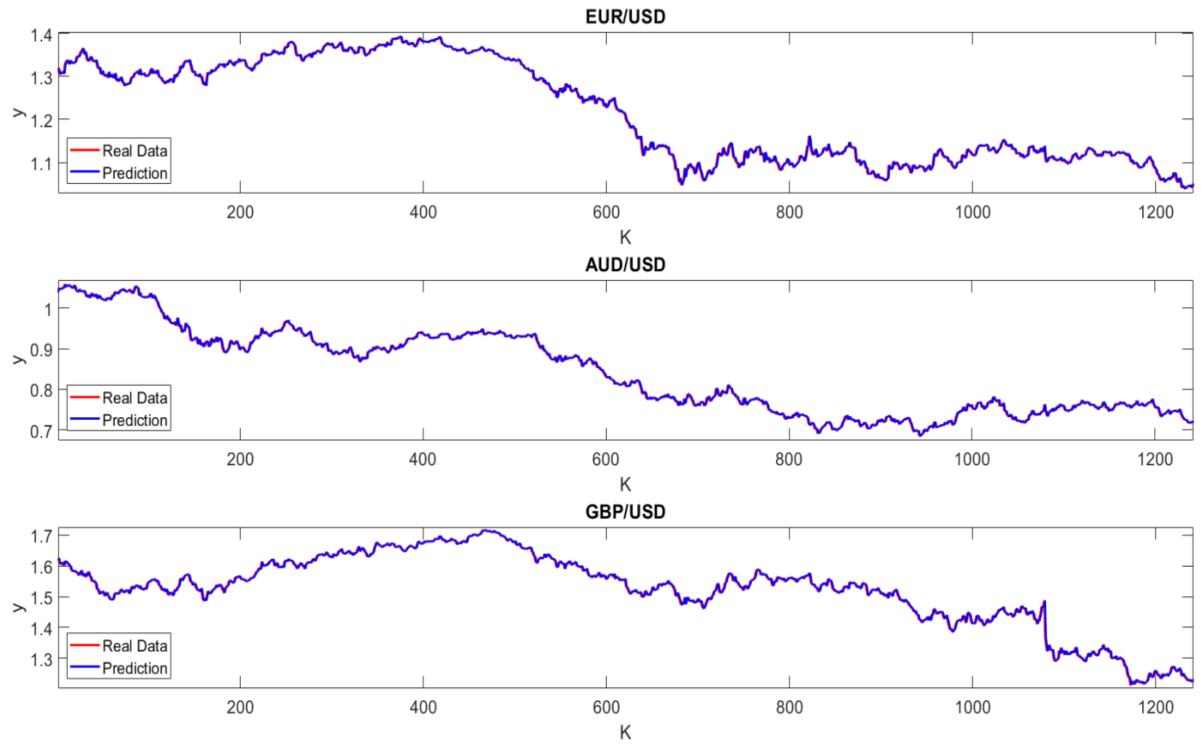


Fig.4. Prediction results on the currency exchange data by SBALMMo

It is also quite interesting to notice that the dummy prediction method actually is able to provide the prediction results in the same level as the SBALMMo. This is because of the stationary nature of this problem, the foreign currency exchange data usually does not change dramatically in a short time. There is no big difference between the close price of one day and the close price of the next day (see Fig. 4). Since all the regression algorithms have inherent approximation errors, when the data is stationary, the approximation errors play a more significant role in their performance.

In order to demonstrate the interpretability of the created fuzzy rules, the rule base obtained after the training processes by the SBALMMo system are tabulated in Table 10, where we can see that the interpretability of the fuzzy rules comes from the prototype-based nature [43]. Prototypes are the most representative data samples, and they represent the local models in the data distribution. The prototypes also make the decision process of the SBALMMo system interpretable. For each newly arrived data sample, the system compares it with all the existing prototypes. The fuzzy rules with prototypes closer to the new data sample have larger firing strengths, which means they contribute more to the system output.

Table 10. Fuzzy rules identified from the training process

	#	Fuzzy Rules
EUR/USD	1	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[0.9285,0.9285,0.9285,0.9285])$ $THEN\ (y=0.0025+0.2497x_{K,1}+0.2493x_{K,2}+0.2497x_{K,3}+0.2495x_{K,4})$
	2	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.3037,1.3038,1.3036,1.3037])$ $THEN\ (y=0.0057+0.2492x_{K,1}+0.2490x_{K,2}+0.2486x_{K,3}+0.2489x_{K,4})$
	3	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.3985,1.3986,1.3983,1.3985])$ $THEN\ (y=0.0053+0.2493x_{K,1}+0.2489x_{K,2}+0.2488x_{K,3}+0.2489x_{K,4})$
	4	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.5108,1.5109,1.5107,1.5108])$ $THEN\ (y=0.0049+0.2494x_{K,1}+0.2488x_{K,2}+0.2491x_{K,3}+0.2490x_{K,4})$
AUD/USD	1	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[0.5560,0.5561,0.5560,0.5560])$ $THEN\ (y=0.0014+0.2493x_{K,1}+0.2497x_{K,2}+0.2495x_{K,3}+0.2500x_{K,4})$
	2	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[0.6525,0.6526,0.6524,0.6525])$ $THEN\ (y=0.0016+0.2492x_{K,1}+0.2497x_{K,2}+0.2494\ x_{K,3}+0.2500x_{K,4})$
	3	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[0.9289,0.9290,0.9288,0.9289])$ $THEN\ (y=0.0017+0.2491x_{K,1}+0.2496x_{K,2}+0.2494x_{K,3}+0.2499x_{K,4})$
	4	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[0.8544,0.8545,0.8543,0.8544])$ $THEN\ (y=0.0017+0.2491x_{K,1}+0.2496x_{K,2}+0.2494x_{K,3}+0.2499x_{K,4})$
	5	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.0331,1.0332,1.0329,1.0331])$ $THEN\ (y=0.0018+0.2490x_{K,1}+0.2496x_{K,2}+0.2495x_{K,3}+0.2500x_{K,4})$
GBP/USD	1	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.6492,1.6493,1.6491,1.6492])$ $THEN\ (y=0.0040+0.2494x_{K,1}+0.2491x_{K,2}+0.2502x_{K,3}+0.2490x_{K,4})$
	2	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.9889,1.9890,1.9888,1.9889])$ $THEN\ (y=0.0039+0.2493x_{K,1}+0.2488x_{K,2}+0.2506x_{K,3}+0.2491x_{K,4})$
	3	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.8643,1.8644,1.8643,1.8643])$ $THEN\ (y=0.0038+0.2493x_{K,1}+0.2488x_{K,2}+0.2505x_{K,3}+0.2492x_{K,4})$
	4	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.4517,1.4518,1.4515,1.4517])$ $THEN\ (y=0.0033+0.2497x_{K,1}+0.2492x_{K,2}+0.2503x_{K,3}+0.2489x_{K,4})$
	5	$IF([x_{K,1},x_{K,2},x_{K,3},x_{K,4}]\sim[1.7686,1.7687,1.7685,1.7686])$ $THEN\ (y=0.0039+0.2494x_{K,1}+0.2490x_{K,2}+0.2503x_{K,3}+0.2491x_{K,4})$

To further evaluate the statistical performance of the proposed SBALMMo on this real-world problem, we apply the five-fold cross-validation method by dividing each dataset into five subsets evenly following the timeline, and each time we use four subsets for training and leave one subset out for validation. We also involve other well-known algorithms for comparison. The statistical results are given in Table 11 in terms of the means and standard deviations of the three measures, namely, RMSE, rule# and t_{exe}. The results obtained by the dummy prediction method are also given in the table as the baseline.

From Table 11 it can be concluded that, in this numerical example, the SBALMMo system outperforms all the comparative algorithms. Despite that the overall training time consumption increased by two to three times compared with the ALMMo system, the self-boosting algorithm significantly reduces the complexity of the SBALMMo system structure.

Table 11. Statistical performance comparison on foreign currency exchange data

	Algorithm	Mean ± Stand Deviation		
		RMSE	rule#	t_{exe} (s)
EUR/USD	**SBALMMo**	**0.0073±0.0019**	**4.3±0.5**	**1.06±0.06**
	ALMMo [6]	0.0074±0.0019	7.5±0.6	0.46±0.02
	FCMMS [36]	0.0163±0.0054	7.4±4.3	0.96±0.25
	AnYa [27]	0.0075±0.0019	2.0±0.0	0.22±0.05
	ESAFIS [35]	0.0074±0.0017	1.8±0.4	1.36±0.34
	SAFIS [17]	0.0669±0.0927	1.8±0.4	0. 36±0.17
	Dummy	0.0072±0.0019		
AUD/USD	**SBALMMo**	**0.0062±0.0019**	**5.3±1.5**	**1.40±0.43**
	ALMMo [6]	0.0063±0.0019	9.5±2.6	0.43±0.02
	FCMMS [36]	0.0265±0.0085	6.4±3.4	0.81±0.11
	AnYa [27]	0.0080±0.0028	2.0±0.0	0.25±0.04
	ESAFIS [35]	0.0063±0.0017	2.0±0.0	1.48±0.39
	SAFIS [17]	0.0513±0.0590	1.8±0.4	0.38±0.02
	Dummy	0.0058±0.0020		
GBP/USD	**SBALMMo**	**0.0088±0.0022**	**6.8±2.2**	**1.38±0.26**
	ALMMo [6]	0.0089±0.0022	12.3±3.5	0.46±0.04
	FCMMS [36]	0.0364±0.0206	11.0±5.3	1.01±0.09
	AnYa [27]	0.0098±0.0022	2.0±0.0	0.24±0.05
	ESAFIS [35]	0.0092±0.0020	1.8±0.4	1.67±0.46
	SAFIS [17]	0.2463±0.0643	2.0±0.0	0.27±0.06
	Dummy	0.0086±0.0023		

5.3. Performance on Nonstationary Regression Problems

Nonstationary regression problems usually have much more frequent and intensive changes in the data pattern, and they are more challenging compared with other types of problems due to their nonlinear, nonstationary, erratic and time-variant behaviour. Since it is of great importance for a learning algorithm to be able to successfully follow the ever-changing data pattern, the nonstationary regression problems are very useful for evaluating this ability.

In this subsection, we consider the following two real-world high frequency trading (HFT) problems:

1) QuantQuote second resolution market dataset [44] and

2) Standard and Poor index dataset [45]

to evaluate the performance of the SBALMMo neuro-fuzzy systems on nonstationary regression problems.

The QuantQuote second resolution market dataset contains tick-by-tick data on all NASDAQ, NYSE and AMEX securities from 1998 to the present moment. The dataset contains 19144 observations with the following attributes:

1) time tag, K,

2) open price, $x_{K,1}$,

3) high price, $x_{K,2}$,

4) low price, $x_{K,3}$ and

5) close price, $x_{K,4}$.

In the following example, we, firstly, investigate the relationship between the size of the data pool, Q and the regression performance of the SBALMMo. In this example, $\Omega_1 = 0$ is fixed and Q is set to be $1000, 2000, 3000, 4000, 5000$ and 6000. We use the current four prices $\boldsymbol{x}_K = [x_{K,1}, x_{K,2}, x_{K,3}, x_{K,4}]^T$ to predict the close price 5 steps ahead, $x_{K+5,4} = f(\boldsymbol{x}_K)$ in a sample-by-sample manner and the performance is tabulated in Table 12 in terms of non-dimensional error index (NDEI), number of fuzzy rules (rule#) and execution time (t_{exe}, in seconds). The average time consumption for each self-boosting is given for evaluation of the computational complexity of the proposed approach. The performance of the ALMMo neuro-fuzzy system is also tabulated in Table 12 as a baseline. The data samples are standardized online before prediction. The NDEI is calculated based on the following equation [6]:

$$NDEI = \sqrt{\frac{\frac{1}{K}\sum_{j=1}^{K} e_j^2}{\sigma_y^2}} \tag{42}$$

where $e_j = \hat{y}_j - y_j$; $\hat{y}_j$ is the system output at the j-th time instance; y_j is the corresponding real output; σ_y is the standard deviation of the real output.

Table 12. Performance on benchmark regression datasets using different value of Q ($\Omega_1 = 0$)

Algorithm	**SBALMMo**						**ALMMo**
Measure	Q						
	1000	2000	3000	4000	5000	6000	
NDEI	0.1301	0.1299	0.1294	0.1292	0.1299	0.1296	0.1339
rule#	7	8	5	5	5	7	6
t_{exe} (s)	14.87	10.81	9.87	9.90	9.44	10.55	2.16
t_{exe} per self-boosting (s)	0.60	0.87	1.17	1.72	2.15	2.64	

As one can see from Table 12, size of the data pool, Q has only very minor influence on the performance of the SBALMMo. Nonetheless, since the SBALMMo achieves the best NDEI and the simplest system structure when $Q = 4000$, we use $Q = 4000$ in the remaining numerical examples of this subsection.

Remark 4: For nonstationary streaming data, a data pool of smaller size allows the SBALMMo to perform self-boosting in real time. However, the performance might be influenced by the smaller data pool as the parameters of the premise (IF) and consequent (THEN) parts of the system converge to a locally minimum solution. Therefore, a trade-off between the performance and computational efficiency needs to be made depending on the preferences/requirements of the problem.

Similar to the previous subsection 5.1, we investigate the relationship between the forgetting factor, Ω_1 and the regression performance of the SBALMMo on the high frequency trading problem. In the following numerical example, we use the same experimental protocol as the previous example, but set $Q = 4000$ and change the value of Ω_1 as $\Omega_1 = -0.02,0,0.02,0.04,0.06,0.08$ and 0.1. The prediction performance is reported in Table 13.

Table 13. Performance comparison using different value of Ω_1 ($Q = 4000$)

Algorithm	**SBALMMo**							**ALMMo**
Measure	Ω_1							
	-0.02	0	0.02	0.04	0.06	0.08	0.10	
NDEI	0.1297	0.1292	0.1295	0.1289	0.1296	0.1296	0.1292	0.1339
rule#	4	5	6	7	7	9	9	6
t_{exe} (s)	6.80	9.90	10.38	14.21	14.95	17.97	20.68	2.16

As one can see from Table 13, a larger value of Ω_1 ($\Omega_1 > 0.04$) would result in lower computational efficiency, more complex system structure. While a very small value of Ω_1 will also influence the regression accuracy. By making a trading-off between the performance and computational efficiency as well as the system complexity, we use $\Omega_1 = 0$ in the remaining numerical examples of this subsection, which is the same as we used before.

In the next example, we conduct the following experiments using the QuantQuote second resolution market dataset:

1) use the four prices of the current step, $\boldsymbol{x}_K$, to predict the close price 5 steps ahead, $x_{K+5,4} = f(\boldsymbol{x}_K)$;

2) use the four prices of the current step, $\boldsymbol{x}_K$, to predict the close price 10 steps ahead, $x_{K+10,4} = f(\boldsymbol{x}_K)$;

3) use the four prices of the current step, $\boldsymbol{x}_K$, to predict the close price 15 steps ahead, $x_{K+15,4} = f(\boldsymbol{x}_K)$;

4) use the four prices of the current step, $\boldsymbol{x}_K$, to predict the close price 20 steps ahead, $x_{K+20,4} = f(\boldsymbol{x}_K)$;

The prediction performance in terms of non-dimensional error index (NDEI), number of fuzzy rules (rule#) and execution time (t_{exe}, in seconds) obtained during the numerical experiments are reported in Table 14.

The prediction results obtained by the SBALMMo and ALMMo neuro-fuzzy systems, respectively, on the third experiment $x_{K+15,4} = f(\boldsymbol{x}_K)$ are depicted in Fig. 5 (a), and the corresponding changes of the number of fuzzy rules are given in Fig. 5(b). We also present the zoomed-in prediction results of six periods:

Period 1: $K \in [4000{,}4500]$,

Period 2: $K \in [5500{,}6000]$,

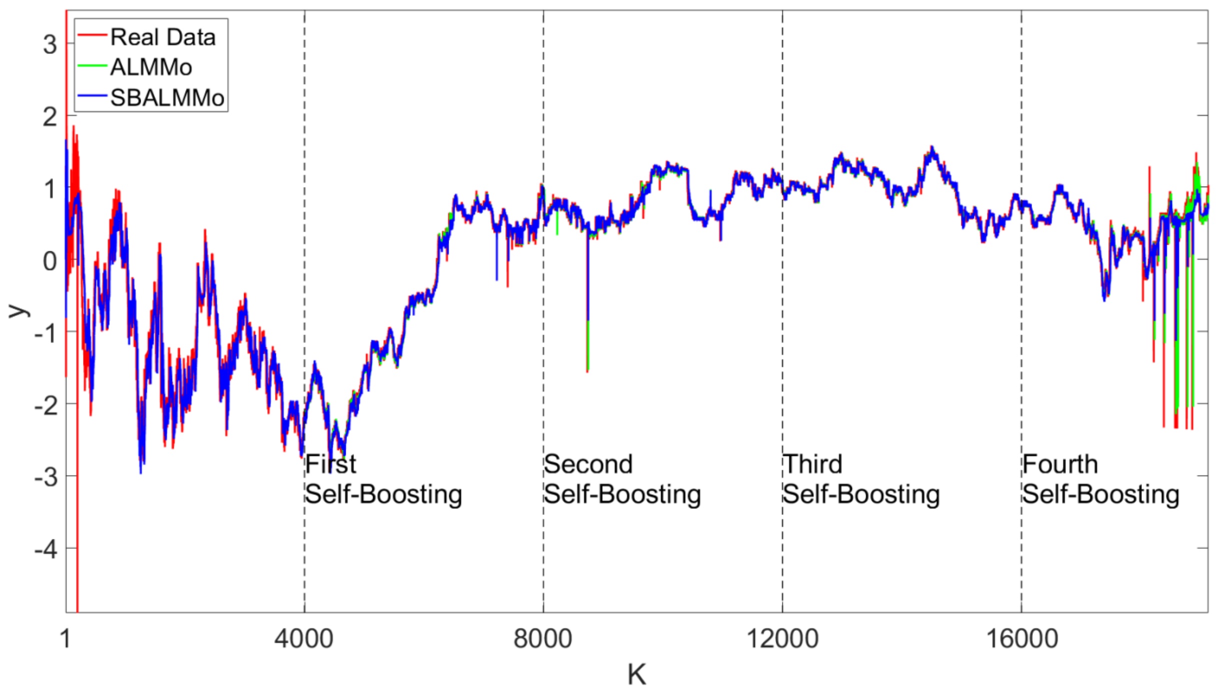
Period 3: $K \in [8000{,}8500]$,

Period 4: $K \in [8500{,}9000]$,

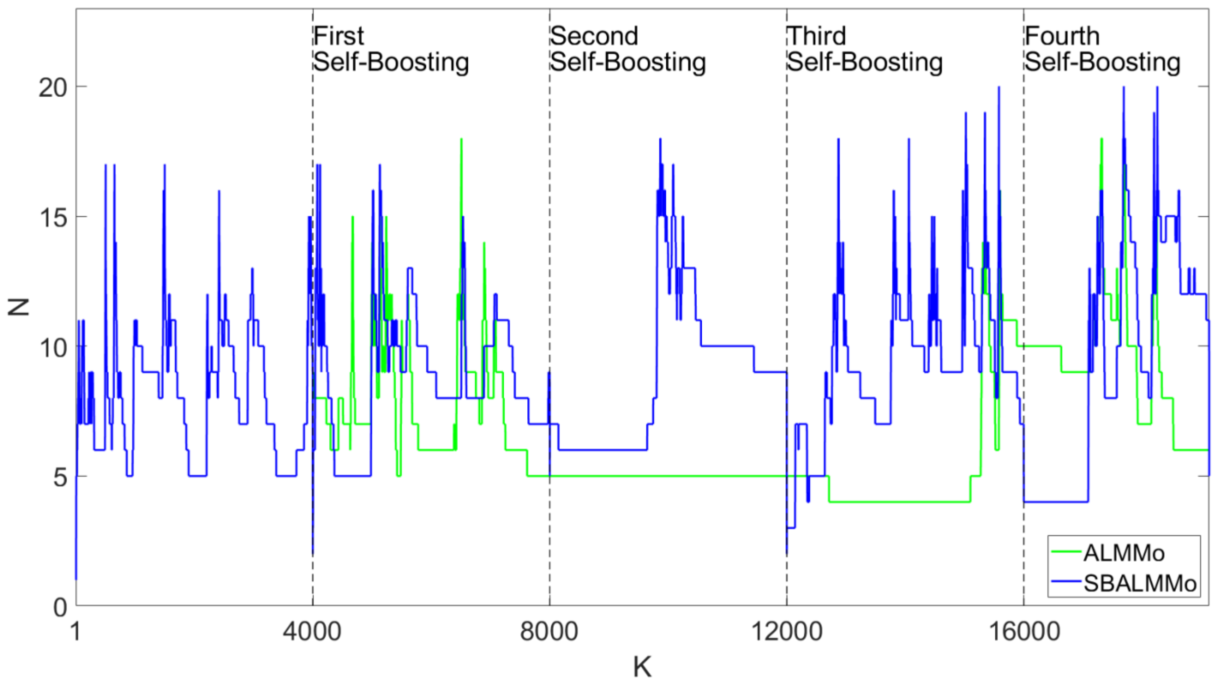
Period 5: $K \in [16000{,}16500]$, and

Period 6: $K \in [18500{,}19000]$,

in Fig. 6 for a better illustration. Furthermore, we depict the curves of the accumulated square error, $\sum_{j=1}^{K} e_j^2$ at each time instance obtained by the ALMMo and SBALMMo during the same numerical experiment in Fig. 7, and the corresponding zoomed-in periods are depicted in Fig. 8. From Figs. 5~8 one can see that the self-boosting algorithm effectively increases the prediction accuracy of the SBALMMo system.



(a) Prediction results



(b) Changes of the number of fuzzy rules

Fig. 5. Prediction performance on QuantQuote second resolution market dataset

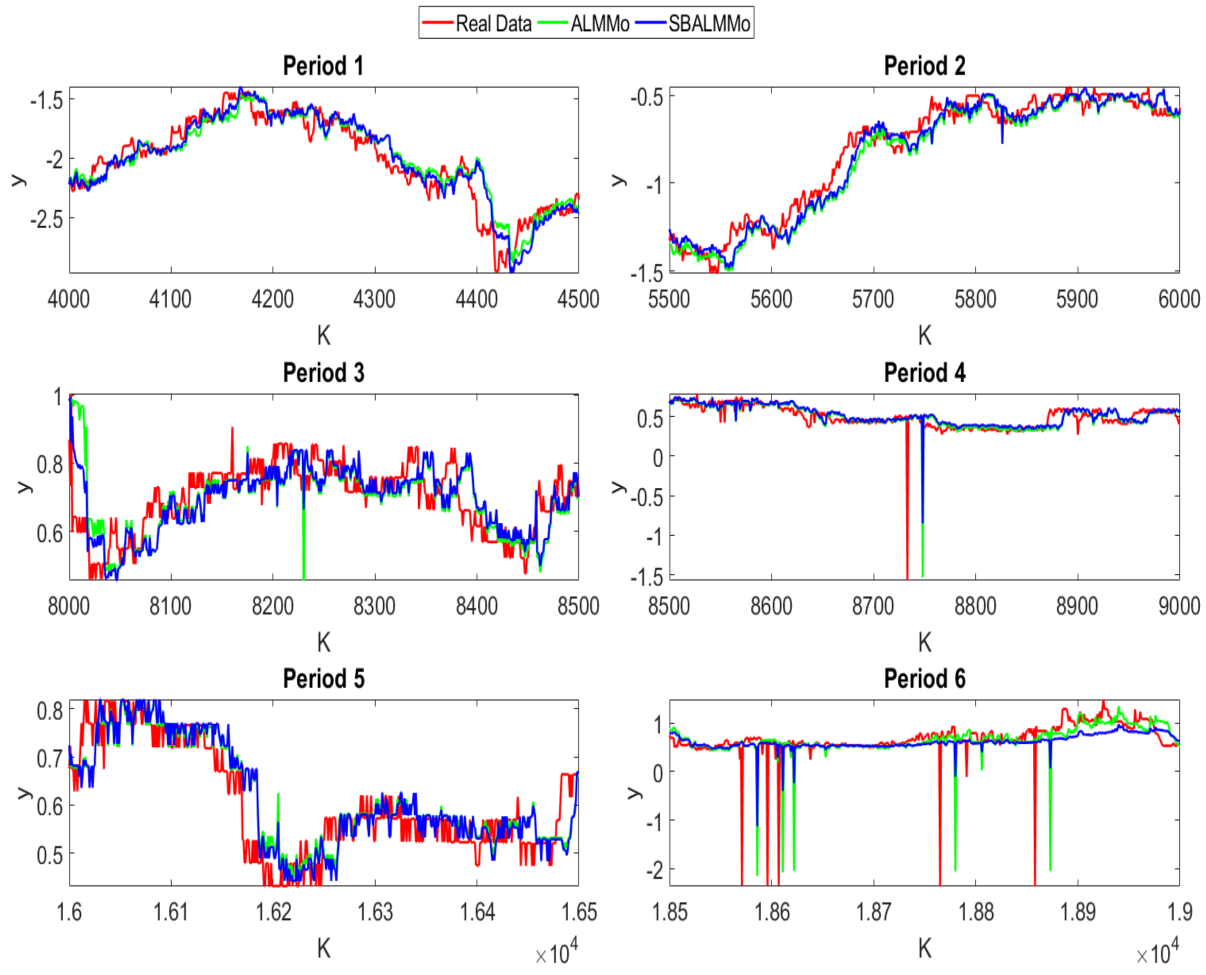


Fig. 6. The zoomed-in prediction results

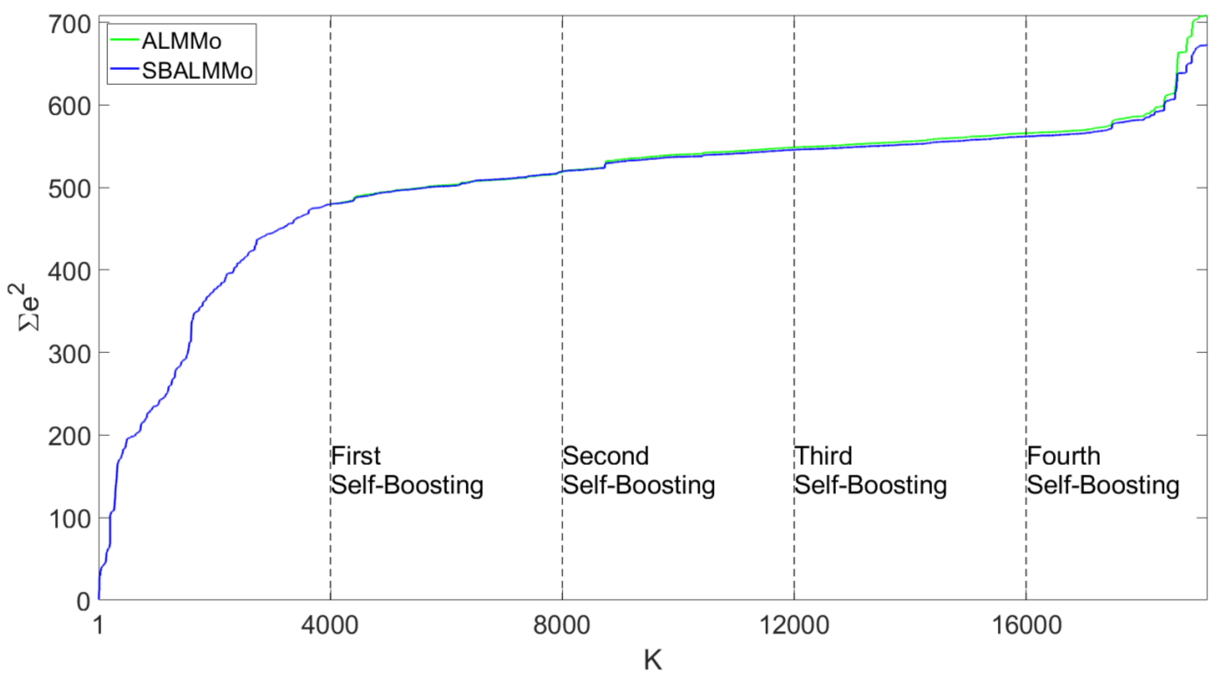


Fig. 7. The curves of the accumulated square error

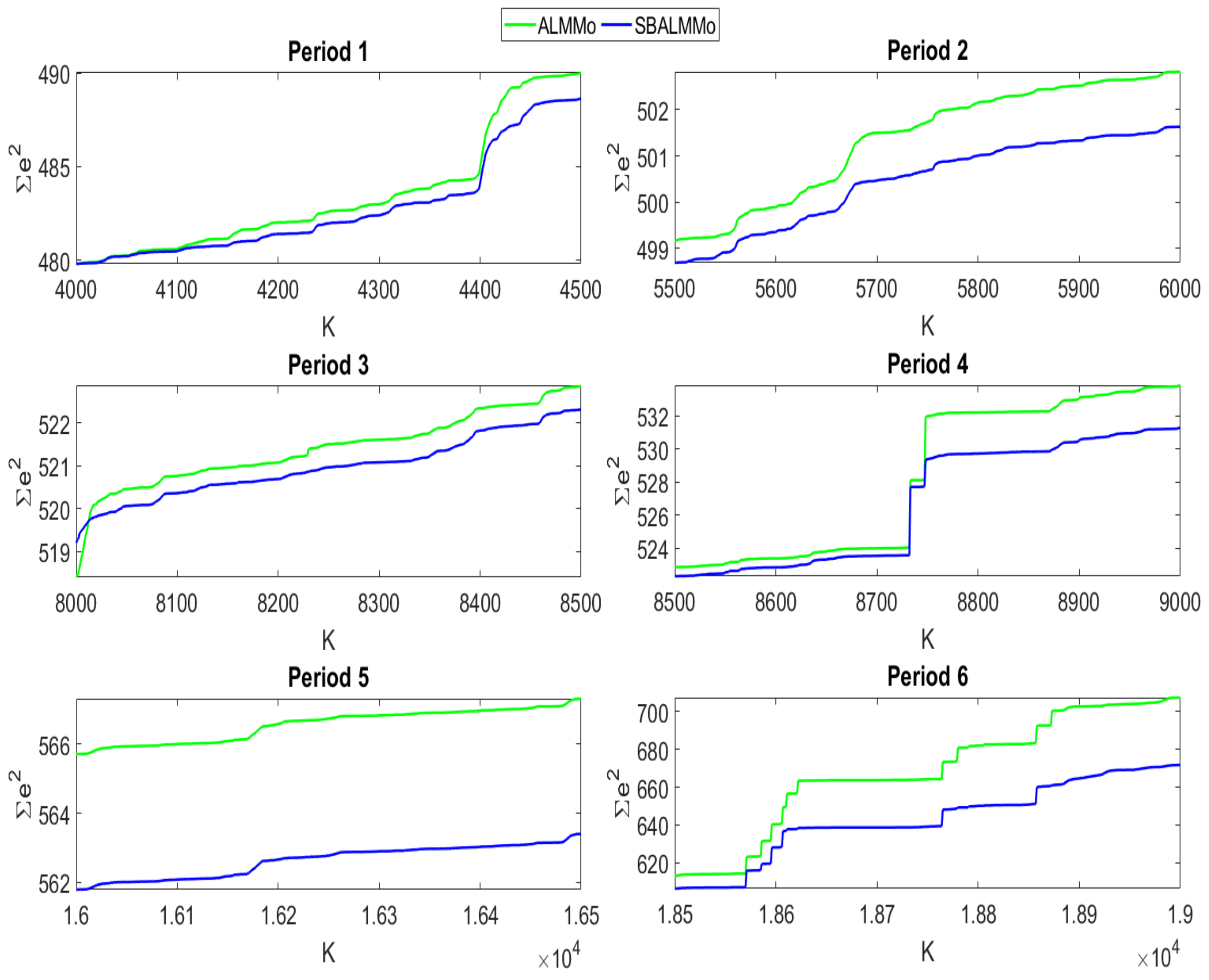


Fig. 8. The six zoomed-in periods of the curves of the accumulated square error

We also involve a number of well-known approaches for comparison under the same experimental protocol, and the results are reported in the same table. Similar to Table 9, the results obtained by dummy prediction method are reported in Table 14 as the baseline, which takes the current closing price as the predicted closing price. From Table 14, one can see that the SBALMMo neuro-fuzzy system is able to outperform all other approaches during the experiments.

Table 14. Performance comparison on high frequency trading dataset

	Algorithm	NDEI	rule#	t_{exe} (s)
Input: $\boldsymbol{x}_K$ Output: $x_{K+5,4}$	**SBALMMo**	**0.1292**	**5**	**9.90**
	ALMMo [6]	0.1339	6	2.11
	FCMMS [36]	0.1322	4	5.31
	AnYa [27]	0.1487	4	3.41
	OLSLR [40]	0.1607		6.67
	SWLSLR [41]	0.1517		0.51
	eTS [3]	0.1405	4	185.28
	SAFIS [17]	0.7692	14	6.45
	Dummy	0.5139		
Input: $\boldsymbol{x}_K$ Output: $x_{K+10,4}$	**SBALMMo**	**0.1575**	**5**	**9.60**
	ALMMo [6]	0.1620	6	2.15
	FCMMS [36]	0.1599	4	5.30
	AnYa [27]	0.1744	4	3.12
	OLSLR [40]	0.1874		6.33
	SWLSLR [41]	0.1625		0.49
	eTS [3]	0.1723	4	182.49
	SAFIS [17]	0.8129	14	8.98
	Dummy	0.5212		
Input: $\boldsymbol{x}_K$ Output: $x_{K+15,4}$	**SBALMMo**	**0.1726**	**5**	**9.41**
	ALMMo [6]	0.1771	6	2.12
	FCMMS [36]	0.1781	4	5.34
	AnYa [27]	0.1931	4	3.43
	OLSLR [40]	0.2056		6.411
	SWLSLR [41]	0.1826		0.50
	eTS [3]	0.1870	4	183.66
	SAFIS [17]	0.6847	16	10.12
	Dummy	0.5251		
Input: $\boldsymbol{x}_K$ Output: $x_{K+20,4}$	**SBALMMo**	**0.1875**	**5**	**9.51**
	ALMMo [6]	0.1920	6	2.34
	FCMMS [36]	0.1945	4	5.97
	AnYa [27]	0.2138	4	3.33
	OLSLR [40]	0.2237		8.92
	SWLSLR [41]	0.2056		0.53
	eTS [3]	0.2130	3	193.13
	SAFIS [17]	0.6940	21	12.73
	Dummy	0.5293		

To further evaluate the statistical performance of the proposed SBALMMo on this nonstationary problem, we apply the five-fold cross-validation method by dividing this dataset into five subsets evenly following the timeline, and each time we use four subsets for training and leave one subset out for validation. We also involve other well-known algorithms for comparison. The results are given in Table 15 in terms of the means and standard deviations of the three measures, namely, RMSE, rule# and t_{exe}.

From Table 15 one can see that in the four numerical experiments, the self-boosting algorithm effectively improves the accuracy of the prediction results obtained by the SBALMMo as well as its robustness. The average NDEI is reduced by around 20%, and the standard deviation of NDEI is reduced by around 50%. Comparing with ALMMo, the system structure complexity of the SBALMMo is also reduced by 33%.

Meanwhile, this comes at the price that the overall computation time of the SBALMMo system is around four times longer than that of the ALMMo system.

Table 15. Statistical Performance comparison on high frequency trading dataset

	Algorithm	**Mean ± Stand Deviation**		
		NDEI	**rule#**	**t_{exe} (s)**
Input: $\boldsymbol{x}_K$ Output: $x_{K+5,4}$	**SBALMMo**	0.3646±0.1274	6.4±0.9	8.36±2.12
	ALMMo [6]	0.4668±0.2634	9.8±2.9	1.73±0.27
	FCMMS [36]	0.4518±0.1266	9.8±2.9	4.54±0.61
	AnYa [27]	0.3697±0.2591	2.4±0.5	1.97±0.16
	ESAFIS [35]	0.9403±0.6731	2.8±0.8	4.96±0.25
	SAFIS [17]	0.9665±0.3755	12.6±2.3	1.84±0.19
	Dummy	1.2441±0.7375		
Input: $\boldsymbol{x}_K$ Output: $x_{K+10,4}$	**SBALMMo**	0.4212±0.1270	6.2±0.8	7.91±1.28
	ALMMo [6]	0.5194±0.2716	9.0±3.4	1.75±0.25
	FCMMS [36]	0.5777±0.3912	7.2±2.8	4.47±0.51
	AnYa [27]	0.4625±0.2350	3.0±0.0	1.91±0.08
	ESAFIS [35]	1.2510±0.7753	2.6±0.9	5.38±0.49
	SAFIS [17]	0.8337±0.4904	10.8±0.8	1.85±0.19
	Dummy	1.2535±0.7303		
Input: $\boldsymbol{x}_K$ Output: $x_{K+15,4}$	**SBALMMo**	0.4549±0.1440	5.4±0.9	7.53±1.24
	ALMMo [6]	0.5567±0.2631	9.6±2.8	1.78±0.31
	FCMMS [36]	0.5514±0.1864	5.0±1.6	4.47±0.20
	AnYa [27]	0.5329±0.2068	3.8±0.4	2.04±0.12
	ESAFIS [35]	1.0190±0.6665	3.2±0.8	5.93±1.10
	SAFIS [17]	1.3247±0.7819	14.4±2.2	2.00±0.29
	Dummy	1.2614±0.7249		
Input: $\boldsymbol{x}_K$ Output: $x_{K+20,4}$	**SBALMMo**	0.5011±0.1377	6.20±0.8	8.30±1.47
	ALMMo [6]	0.6156±0.2741	9.6±2.8	2.03±0.50
	FCMMS [36]	0.7957±0.4396	7.2±1.6	4.61±0.84
	AnYa [27]	0.6419±0.2987	3.8±0.4	2.08±0.05
	ESAFIS [35]	0.6593±0.2914	3.4±0.9	6.08±0.91
	SAFIS [17]	1.4510±0.8920	17.0±5.3	2.75±1.14
	Dummy	1.2693± 0.7201		

The Standard and Poor index dataset contains 14893 observations acquired from January 3, 1950 to March 12, 2009. The input and output relationship of the system is governed by $x_{K+1} = f([x_K, x_{K-1}, x_{K-2}, x_{K-3}, x_{K-4}]^T)$. The online prediction performance on the Standard and Poor index dataset obtained by the regression algorithms is tabulated in Table 16 in terms of NDEI and rule#. Similarly, the dummy prediction method, which uses the current observation as the predicted one, namely, $x_{K+1} \leftarrow x_K$, is used as the baseline approach in the comparison. From Table 16 we can see that both, the SBALMMo and ALMMo achieve the best performance, but the SBALMMo has a simpler system structure than ALMMo.

Table 16. Performance comparison on the Standard and Poor dataset

Algorithm	**NDEI**	**rule#**
SBALMMo	**0.013**	**7**
ALMMo [6]	0.013	8
AnYa [27]	0.018	11
OLSLR [40]	0.020	
SWLSLR [41]	0.018	
FCMMS [36]	0.014	5

eTS [3]	0.015	14
DENFIS [2]	0.020	6
SAFIS [17]	0.209	6
EFuNN [39]	0.154	114.3
SeroFAM [38]	0.027	29
Simpl_eTS [13]	0.045	7
PANFIS [16]	0.014	4
Dummy	0.029	

To further evaluate the statistical performance of the proposed SBALMMo on this problem, we firstly divide this dataset into five subsets evenly following the timeline. Then, we assemble the five subsets in a random order into a new sequence and perform the online prediction using the same experimental protocol as used in the previous experiment. The statistical results are given in Table 17 after 10 Monte-Carlo experiments in terms of the means and standard deviations of the three measures, namely, RMSE, rule# and t_{exe}. We also involve other well-known algorithms for comparison.

Table 17. Performance comparison on the Standard and Poor dataset

Algorithm	**Mean ± Stand Deviation**		
	NDEI	**rule#**	**t_{exe} (s)**
SBALMMo	0.0145±0.0038	5.8±3.5	35.17±9.70
ALMMo [6]	0.0148±0.0039	8.6±3.5	2.10±0.24
FCMMS [36]	0.0178±0.0062	6.0±4.4	4.10±0.30
AnYa [27]	0.0290±0.0048	11.0±3.1	3.21±0.12
ESAFIS [35]	0.1815±0.2043	1.2±0.4	4.50±0.97
SAFIS [17]	0.1878±0.0731	5.9±28	1.62±0.56
Dummy	0.0287±0.0000		

From Table 17 one can see that, there is a marginal improvement in the NDEI in the proposed SBALMMo compared with the ALMMo. The system structure complexity of the SBALMMo is reduced by around 33%, but the overall execution time is increased 17 times.

5.4. Discussions

Form the numerical examples provided in this section one can see that the proposed self-boosting algorithm is able to effectively optimize the system structure and meta-parameters in few iterations. In comparison with alterative techniques, the unique advantages that the self-boosting algorithm brings to the proposed SBALMMo neuro fuzzy system include:

1) improving the accuracy of the regression and prediction results;

2) improving the robustness of the system performance;

3) reducing the system structure complexity.

The proposed SBALMMo is able to produce better results in different benchmark datasets and real-world problems compared with the "state-of-the-art" approaches. In addition, it is able to perform prediction on nonstationary problems with higher accuracy, and is capable of performing self-boosting in real time. Therefore, one may conclude that the SBALMMo can serve as a strong alternative to other approaches.

The practical implementation of the self-boosting algorithm is very flexible, and can be triggered based on different criteria. The main aim of this paper is to deliver the concepts and the principles, and, thus, we only applied the general implementations to the numerical examples. Alternative criteria for triggering the self-boosting can be, for example, when the accumulated errors or the error between the system output and the reference output exceeds a certain threshold [24]. However, this requires a deeper study since the accuracy of a learning algorithm in terms of regression error is highly dependent on the problem itself (one can see from the previous numerical examples), and it is out of the scope of this paper.

The proposed self-boosting algorithm is a generic optimization algorithm that can be used in other neuro-fuzzy systems of AnYa type with FWRLS consequent parameter updating algorithm. This work touches the very foundation of EISs.

6. Conclusion

In this paper, we conducted a deep analysis on the optimality of the premise (IF) and consequent (THEN) parts of the recently introduced ALMMo neuro-fuzzy system, and further proposed a self-boosting mechanism that consists of the structure- and parameter-optimization algorithm to enable the ALMMo system to self-optimize its system structure and meta-parameters. We named the upgraded ALMMo learning system as self-boosting autonomous learning multi-model (SBALMMo) neuro-fuzzy systems. Numerical examples based on benchmark datasets and real-world problems demonstrate the validity of the concepts and principles. It shows the strong potential of the SBALMMo for real applications.

As future work, we will further improve SBALMMo neuro-fuzzy systems by employing more intelligent triggering mechanisms for the self-boosting operation, i.e. based on the accumulated errors. We will also apply the self-boosting algorithm to other types of neuro-fuzzy systems.

7. References

[1] P. Angelov and R. Buswell, "Evolving rule-based models: a tool for intelligent adaption," in *IFSA World Congress and 20th NAFIPS International Conference*, 2001, pp. 1062–1067.

[2] N. K. Kasabov and Q. Song, "DENFIS : Dynamic Evolving Neural-Fuzzy Inference System and Its Application for Time-Series Prediction," *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 2, pp. 144–154, 2002.

[3] P. P. Angelov and D. P. Filev, "An approach to online identification of Takagi-Sugeno fuzzy models," *IEEE Trans. Syst. Man, Cybern. - Part B Cybern.*, vol. 34, no. 1, pp. 484–498, 2004.

[4] P. Angelov, *Autonomous learning systems: from data streams to knowledge in real time*. John Wiley & Sons, Ltd., 2012.

[5] L. Maciel, R. Ballini, and F. Gomide, "Evolving possibilistic fuzzy modeling for realized volatility forecasting with jumps," *IEEE Trans. Fuzzy Syst.*, vol. 25, no. 2, pp. 302–314, 2017.

[6] P. P. Angelov, X. Gu, and J. C. Principe, "Autonomous learning multi-model systems from data streams," *IEEE Trans. Fuzzy Syst.*, vol. 26, no. 4, pp. 2213–2224, 2018.

[7] A. B. Asghar and X. Liu, "Adaptive neuro-fuzzy algorithm to estimate effective wind speed and optimal rotor speed for variable-speed wind turbine," *Neurocomputing*, vol. 272, pp. 495–504, 2018.

[8] A. B. Asghar and X. Liu, "Estimation of wind speed probability distribution and wind energy potential using adaptive neuro-fuzzy methodology," *Neurocomputing*, vol. 287, pp. 58–67, 2018.

[9] C. F. Juang and C. T. Lin, "An on-line self-constructing neural fuzzy inference network and its applications," *IEEE Trans. Fuzzy Syst.*, vol. 6, no. 1, pp. 12–32, 1998.

[10] E. H. Mamdani, "Application of fuzzy algorithms for control of simple dynamic plant," *Proc. Inst. Electr. Eng.*, vol. 121, no. 12, p. 1585, 1974.

[11] J. S. R. Jang, "ANFIS: adaptive-network-based fuzzy inference system," *IEEE Trans. Syst. Man Cybern.*, vol. 23, no. 3, pp. 665–685, 1993.

[12] E. Lughofer and P. Angelov, "Handling drifts and shifts in on-line data streams with evolving fuzzy systems," *Appl. Soft Comput.*, vol. 11, no. 2, pp. 2057–2068, 2011.

[13] P. Angelov and D. Filev, "Simpl _ eTS : A simplified method for learning evolving Takagi-Sugeno fuzzy models," in *IEEE International Conference on Fuzzy Systems*, 2005, pp. 1068–1073.

[14] P. Angelov, E. Lughofer, and X. Zhou, "Evolving fuzzy classifiers using different model architectures," *Fuzzy Sets Syst.*, vol. 159, no. 23, pp. 3160–3182, 2008.

[15] R. D. Baruah, P. P. Angelov, and J. Andreu, "Simpl _ eClass : simplified potential-free evolving fuzzy rule-based classifiers," in *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*,

2011, pp. 2249–2254.

[16] M. Pratama, S. G. Anavatti, P. P. Angelov, and E. Lughofer, "PANFIS : a novel incremental learning machine," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 25, no. 1, pp. 55–68, 2014.

[17] H. J. Rong, N. Sundararajan, G. Bin Huang, and P. Saratchandran, "Sequential adaptive fuzzy inference system (SAFIS) for nonlinear system identification and prediction," *Fuzzy Sets Syst.*, vol. 157, no. 9, pp. 1260–1275, 2006.

[18] R. Bao, H. Rong, P. P. Angelov, B. Chen, and P. K. Wong, "Correntropy-based evolving fuzzy neural system," *IEEE Trans. Fuzzy Syst.*, vol. 26, no. 3, pp. 1324–1338, 2018.

[19] M. Pratama, S. G. Anavatti, and E. Lughofer, "Genefis: toward an effective localist network," *IEEE Trans. Fuzzy Syst.*, vol. 22, no. 3, pp. 547–562, 2014.

[20] P. Angelov, "Evolving takagi-sugeno fuzzy systems from streaming data (eTS+)," in *Evolving intelligent systems: methodology and applications*, John Wiley & Sons., 2010.

[21] E. Lughofer, *Evolving fuzzy systems-methodologies, advanced concepts and applications*. Berlin: Springer, 2011.

[22] E. Lughofer, "Evolving fuzzy systems—fundamentals, reliability, interpretability, useability, applications," in *Handbook on Computational Intelligence*, P. Angelov, Ed. New York: World Scientific, 2016, pp. 67–135.

[23] R. M. Johnstone, C. Richard Johnson, R. R. Bitmead, and B. D. O. Anderson, "Exponential convergence of recursive least squares with exponential forgetting factor," *Syst. Control Lett.*, vol. 2, no. 2, pp. 77–82, 1982.

[24] D. Ge and X. J. Zeng, "Learning evolving T-S fuzzy systems with both local and global accuracy - a local online optimization approach," *Appl. Soft Comput.*, vol. 86, pp. 795–810, 2018.

[25] H.-J. Rong, P. Angelov, X. Gu, and J.-M. Bai, "Stability of evolving fuzzy systems based on data clouds," *IEEE Trans. Fuzzy Syst.*, vol. 26, no. 5, pp. 2774–2784, 2018.

[26] G. Leng, G. Prasad, and T. M. McGinnity, "An on-line algorithm for creating self-organizing fuzzy neural networks," *Neural Networks*, vol. 17, no. 10, pp. 1477–1493, 2004.

[27] P. Angelov and R. Yager, "A new type of simplified fuzzy rule-based system," *Int. J. Gen. Syst.*, vol. 41, no. 2, pp. 163–185, 2011.

[28] P. Angelov and X. Zhou, "Evolving fuzzy-rule based classifiers from data streams," *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 6, pp. 1462–1474, 2008.

[29] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its applications to modeling and control," *IEEE Trans. Syst. Man. Cybern.*, vol. 15, no. 1, pp. 116–132, 1985.

[30] W. H. Ho and J. H. Chou, "Design of optimal controllers for Takagi-Sugeno fuzzy-model-based systems," *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans*, vol. 37, no. 3, pp. 329–339, 2007.

[31] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: a generalized convergence theorem and characterization of local optimality," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-6, no. 1, pp. 81–87, 1984.

[32] X. Gu, P. P. Angelov, and H.-J. Rong, "Local optimality of zero-order autonomous learning neuro-fuzzy systems," *Under Rev.*, 2018.

[33] M. R. Azimi-Sadjadi and R. J. Liou, "Fast learning process of multilayer neural networks using recursive least squares method," *IEEE Trans. Signal Process.*, vol. 40, no. 2, pp. 446–450, 1992.

[34] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *J. Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, 1963.

[35] H. J. Rong, N. Sundararajan, G. Bin Huang, and G. S. Zhao, "Extended sequential adaptive fuzzy inference system for classification problems," *Evol. Syst.*, vol. 2, no. 2, pp. 71–82, 2011.

[36] P. Angelov, "Fuzzily connected multimodel systems evolving autonomously from data streams," *IEEE Trans. Syst. Man, Cybern. Part B Cybern.*, vol. 41, no. 4, pp. 898–910, 2011.

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

[38] J. Tan and C. Quek, "A BCM theory of meta-plasticity for online self-reorganizing fuzzy-associative learning," *IEEE Trans. Neural Networks*, vol. 21, no. 6, pp. 985–1003, 2010.

[39] N. Kasabov, "Evolving fuzzy neural networks for supervised/unsupervised online knowledge-based learning," *IEEE Trans. Syst. Man, Cybern. Part B*, vol. 31, no. 6, pp. 902–918, 2001.

[40] C. Nadungodage, Y. Xia, F. Li, J. Lee, and J. Ge, "StreamFitter: A real time linear regression analysis system for continuous data streams," in *International Conference on Database Systems for Advanced Applications*, 2011, pp. 458–461.

[41] K. Tschumitschew and F. Klawonn, "Effects of drift and noise on the optimal sliding window size for data stream regression models," *Commun. Stat. - Theory Methods*, vol. 46, no. 10, pp. 5109–5132, 2017.

[42] "Foreign Currency Exchange Data," *http://www.histdata.com/*. .

[43] P. Angelov and X. Gu, *Empirical approach to machine learning*. Springer International Publishing, 2018.

[44] "QuantQuote Second Resolution Market Database," *https://quantquote.com/historical-stock-data*. .

[45] "Standard and poor (S&P) index data," *https://finance.yahoo.com/quote/%5EGSPC/history?p=%5EGSPC*. .