

An Online Predictor Model as Adaptive Linear model and Evolving Takagi-Sugeno Model

Ahmad Kalhor, Babak N. Araabi and Caro Lucas¹

Abstract—In this paper, we suggest an online predictor model to follow the variations of a process with significant uncertainties. This model is defined as an adaptive linear model initially; however, it starts to evolve into a TS model and continues the process while the modeling error is higher than an adaptive threshold. The model habitually returns to an adaptive linear model when the modeling error becomes lower than the adaptive threshold. Evolving from a linear model to a nonlinear one and then returning to a linear model is performed through specially designed split and merge procedures. Finally, a real-world case study is conducted: prediction of monthly number of sunspots demonstrates the capabilities of the proposed approach.

I. INTRODUCTION

Modeling processes with time varying behaviors, where stationary and fixed models are not suitable, is a demanding task. Online modeling is a fitting solution, where structure and parameters of model can be changed adaptively with respect to the real-time process operation.

Due to their interpretable and flexible structures, Takagi-Sugeno (TS) fuzzy models are suitable tools for online modeling, [1] and [2]: To adapt a TS fuzzy model for time-varying behaviors of a process, fuzzy rules can be easily increased or decreased.

An adaptive online learning algorithm named Dynamic Evolving Neural-Fuzzy Inference System (DENFIS) is proposed in [3]. In [4], a hierarchical on-line self-organizing learning algorithm is proposed to identify a TSK fuzzy model. An online identification approach of the TS model is introduced in [5], where a subtractive clustering and a potential concept are used to define the antecedent parts of the rules. In [6], an online identification of fuzzy neural networks is proposed, which is then applied to NARMAX time series prediction in two different feed-forward and recurrent models. Another online identification approach for the TS model is introduced in [7] where the input space of system is partitioned into a lattice, and local linear models are validated to keep the structure intact. In [8], an approach to online identification of fuzzy rules of an extended TS

fuzzy model is proposed. Fuzzy rule structures are formed through an incremental clustering approach and parameters are updated, recursively, through a non-iterative learning process. An online growing and pruning strategy is proposed in [9] based on contribution of fuzzy rules to the modeling accuracy. In [10], a modeling of switching systems is proposed in which parameters and the order of systems can change when the system switches. In this approach, the local functioning modes were tuned through using an online clustering method. In [11], a fast online neuro-fuzzy modeling is proposed. In this paper, an online fuzzy clustering is introduced which is independent of common spatial features.

In many online modeling approaches, such as the above mentioned ones, the initial defined model evolves gradually to follow the variations of a process. This strategy often makes favorable results in modeling the processes with no significant uncertainties. Insufficient data points, abrupt time-varying behaviors, undetermined effective inputs and un-modeled dynamics of a process accounts for significant uncertainties in an online modeling procedure. In these cases, the model diverges from the process, since the model inertly changes structures and lacks the necessary agility to follow the resulting erratic variations in the process.

Here, we take advantage of linear models as an easy-to-adapt structure to follow the erratic variations occurring in a nonlinear process, whereas the required complexity is provided by allowing each linear model to evolve transiently to a nonlinear one. The proposed Adaptive Habitually Linear and Transiently Nonlinear Model (AHLTNM) can follow fast and significant structural variations in the processes with significant uncertainties. The resulting model is linear until the difference between the model and the process increases beyond an adaptive bound. In this situation, the model evolves into a TS nonlinear model to localize the effect of the errors. The complexity of the nonlinear model increases gradually by adding more rules through splitting existing ones. Splitting is maintained, until the output error complies with the adaptive bound. While the output error is mitigated, the AHLTNM starts to reduce the complexity by merging the existing rules. The AHLTNM, thus, tends to return to an adaptive linear model. As a result, the merging of rules continues as long as error bounds are satisfied. The process of evolving from an adaptive linear model to an adaptive TS nonlinear model and then returning to another linear model is not necessarily a monotonic one. That is, the split and merge processes may switch a few

The authors are with the Control and Intelligent Processing Center of Excellence, School of Electrical and Computer Engineering, University of Tehran, P.O. Box 14395/515, Tehran 14395, Iran (emails: akalhor@ut.ac.ir, lucas@ipm.ir, araabi@ut.ac.ir)

Caro Lucas and Babak N. Araabi are also with the School of Cognitive Sciences, Institute for Studies in Theoretical Physics and Mathematics, Tehran, Iran.

times.

The rest of this paper is organized as follows: Section II describes the AHLTNM along with its learning algorithm. Section III the proposed HLTNM is used to model a real-world process: prediction of monthly sunspots number. Also, the performance of the HLTNM is compared with two other online modeling approaches. The paper is concluded in Section IV.

II. THE STRUCTURE OF AHLTNM AND ITS LEARNING PROCEDURES

A. The Structure of the model in AHLTNM

Each transiently nonlinear model in AHLTNM is described as a TS model:

$$\text{if } \mathbf{x} \text{ is } A_j \text{ then } y^j = \mathbf{z}^T \mathbf{w}_j \quad j = 1, 2, \dots, M \quad (1)$$

where $\mathbf{x} \in \mathbf{R}^n$ is the input vector and $\mathbf{z}^T = [1 \ \mathbf{x}^T]$, y^j is the local output of j th LLM, and $\mathbf{w}_j \in \mathbf{R}^{n+1}$ is the linear parameters vector of j th LLM (more generally, linear parameters of j th rule). The validity function of the j th rule is A_j , which is a fuzzy subset of the input space $\mathbf{R}^n$. The scalar output of model is computed as follows:

$$\hat{y} = \sum_{j=1}^M y^j \phi_j(\mathbf{x}), \quad \phi_j(\mathbf{x}) = \frac{A_j(\mathbf{x})}{\sum_{h=1}^M A_h(\mathbf{x})} \quad (2)$$

where $\phi_j(\mathbf{x})$ denotes the membership function of normalized A_j . In AHLTNM, A_j is defined as the sum of N_j standard Gaussian functions:

$$A_j(\mathbf{x}) = \sum_{i=1}^{N_j} A_{ji}(\mathbf{x}) \quad (3)$$

$$A_{ji}(\mathbf{x}) = \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{c}_{ji})^T \mathbf{S}_{ji}^{-1}(\mathbf{x} - \mathbf{c}_{ji})\right)$$

where $\mathbf{c}_{ji} \in \mathbf{R}^n$ and $\mathbf{S}_{ji} \in \mathbf{R}^{n \times n}$ denote the mean and the diagonal covariance matrix of Gaussian function A_{ji} , respectively.

A Typical contour of A_{ji} , as a standard Gaussian function, is a hyper-ellipse; therefore, A_{ji} can be defined by the center ($\mathbf{c}_{ji}$) and the diameter vector ($\mathbf{L}_{ji}$) of a contour

$$A_{ji}(\mathbf{x}) = \exp\left(-\sum_{k=1}^n (\gamma(\mathbf{x}(k) - \mathbf{c}_{ji}(k)) / \mathbf{L}_{ji}(k))^2\right) \quad (4)$$

It is easy to see that $(\gamma^{-1} \mathbf{L}_{ji}(k))^2$ equals k th diagonal element of diagonal covariance matrix $\mathbf{S}_{ji}$, where γ^{-1} is an interpolation coefficient for membership functions, chosen

typically 1/3 to make a reasonable interpolation [12]. Now, each $(\mathbf{L}_{ji}, \mathbf{c}_{ji})$ pair defines a standard hyper-rectangle b_{ji} , centered at $\mathbf{c}_{ji}$ as shown in Fig. 1.

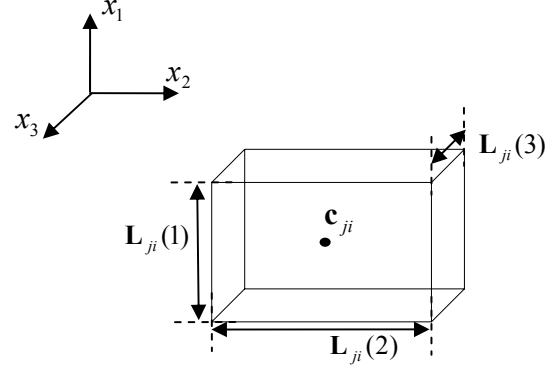


Fig. 1. A three-dimensional scheme for a standard hyper rectangle b_{ji} with center $\mathbf{c}_{ji}$ and length $\mathbf{L}_{ji}$ assigned to A_{ji}

A cluster $\mathbf{C}_j$ is defined as the union of N_j standard hyper-rectangles b_{ji} , $i = 1, 2, \dots, N_j$. As we will see in Section B, these b_{ji} s are disjoint subsets of $\mathbf{R}^n$ which make a single connected cluster, $\mathbf{C}_j$. Each crisp cluster $\mathbf{C}_j$, is associated with a membership function $\phi_j(\cdot)$. Note that $\phi_j(\cdot)$ s are not disjoint and each one constitutes a fuzzy partition on $\mathbf{R}^n$. Fig. 2 shows a transiently nonlinear model in AHLTNM as a TS-type model.

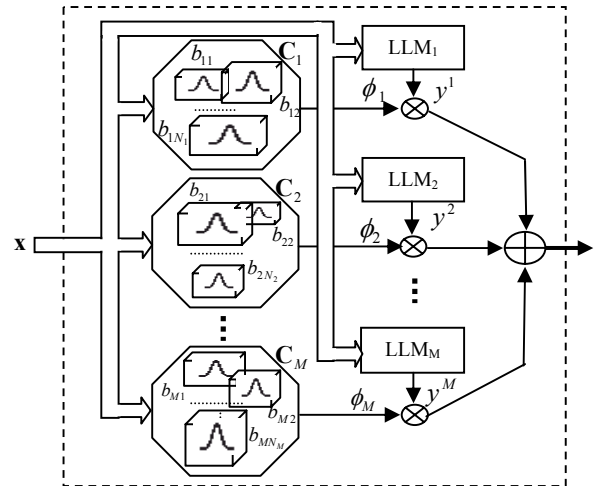


Fig. 2. A block diagram of the transiently nonlinear structure used in AHLTNM. The overall output of model is a weighted sum of local outputs of LLMs, weighted by validity function values.

B. An overview of the AHLTNM learning

Before going on into the details of our learning algorithm for AHLTNM, at first, we give an overview of the proposed algorithm. Assume, at one stage of our online

modeling, the AHLTNM consists of M rules. The newly observed input-output data is $(\mathbf{x}_t, y_t)$. We also consider the $y_{t-\tau}$, where $\tau \geq 1$ is a time delay which the output of system becomes observable after. Since, in the online prediction procedure, the observed data are obtained one by one, it is better to update only the part of the model that is exposed to the current data point directly. Hence, it is assumed that the l th rule, whose validity function in $\mathbf{x}_t$ has the biggest value, is updated and becomes the candidate for more changes.

$$l = \arg \max_j (A_j(\mathbf{x}_t)), j \in \{1, 2, \dots, M\} \quad (5)$$

The main parts of the online learning algorithm are:

- 1- Compute the initial modeling error ($\tilde{e}_t$) as the difference between the output of model, $\hat{y}_t$, and the actual observed output, y_t .
- 2- Use $\tilde{e}_t$ to update the linear parameters of the LLM corresponding to the l th rule with maximum validation function, $A_l(\mathbf{x}_t)$.
- 3- Re-compute the modeling error (e_t) after updating the linear parameters.
- 4- Make an adaptive threshold, $E_t = \eta |y_t - y_{t-\tau}|$, where $\eta > 0$ is introduced to characterize the uncertainty of the process and is adjusted by a heuristic search method.
- 5- If $|e_t| \geq E_t$ and $M < M_{\max}$, run the split procedure, where $M_{\max}$ denotes the maximum number of rules allowed by the designer. Here, the l th rule is split into two rules.
- 6- If $|e_t| < E_t$ and $M > 1$, run the merge procedure. Here, the l th rule is merged with the rule nearest to it, and a combined rule is formed.

The split and merge procedures are aimed to change the structure of the model. The process—and as a result, the model—is assumed to be habitually linear ($M = 1$). Adaptation of the linear model parameters provides some degree of flexibility. The model evolves by splitting into a nonlinear one ($M > 1$) only if the local error cannot be mitigated by parameter adaptation. This is a structural change. However, the nonlinear structure is a transient one which facilitates copies with nonlinear transition in the process. The merging procedure is adopted as a conservative operation that tends to bring the structure of the model back into the adaptive linear structure. Thus, the model tends to simplify the structure and eventually shape back into an adaptive linear model.

C. Details of the AHLTNM learning algorithm

In this section, we provide mathematical and computational details of the procedures introduced in

Section B at different steps of the AHLTNM learning algorithm.

C.1. Updating the linear parameters

The parameters of the l th rule are updated through the Recursive Least Squares (RLS) algorithm applied to weighted input-output data, which are defined as follows:

$$\begin{aligned} \mathbf{z}_{il} &= \mathbf{z}_t (\phi_l(\mathbf{x}_t))^{0.5} \\ y_{il} &= y_t (\phi_l(\mathbf{x}_t))^{0.5} \end{aligned} \quad (6)$$

To update $\mathbf{w}_l$, last local linear parameters vector $\mathbf{w}_l^{t-1}$ is used as follows:

$$\begin{aligned} \mathbf{w}_l^t &= \mathbf{w}_l^{t-1} - \mathbf{P}_l^{t-1} \mathbf{z}_{il} e_{il} \\ e_{il} &= \tilde{e}_t (\phi_l(\mathbf{x}_t))^{0.5}, \quad \tilde{e}_t = y_t - \hat{y}_t \\ \mathbf{P}_l^t &= \mathbf{P}_l^{t-1} - \frac{\mathbf{P}_l^{t-1} \mathbf{z}_{il} \mathbf{z}_{il}^T \mathbf{P}_l^{t-1}}{1 + \mathbf{z}_{il}^T \mathbf{P}_l^{t-1} \mathbf{z}_{il}} \end{aligned} \quad (7)$$

C.2. Split Procedure

There are two different cases for split operation on l th rule:

a) The cluster $\mathbf{C}_l$ of l th rule consists of only one hyper-rectangular block ($N_l = 1, \mathbf{C}_l = \{b_l\}$)

In this case, the $\mathbf{C}_l$ is halved into two clusters through orthogonal-to-axis division of b_l , where both new clusters consist of equal hyper-rectangular blocks. As a result, there are n different pairs of clusters corresponding with n input dimensions, and n orthogonal divisions. In AHLTNM, among n possible splits, we choose the one that better localizes $\mathbf{x}_t$ and its corresponding error. Hence, among all possible divisions ($i = 1, 2, \dots, n$) the division whose one of two resulting hyper-rectangles is closest to $\mathbf{x}_t$ is selected:

$$\begin{aligned} i^* &= \arg \min_i (D_i^l) \\ D_i^l &= \min(\|\mathbf{x}_t - \mathbf{c}_{r1}^i\|, \|\mathbf{x}_t - \mathbf{c}_{s1}^i\|), \quad i = 1, 2, \dots, n \end{aligned} \quad (8)$$

Where $\mathbf{c}_{r1}^i$ and $\mathbf{c}_{s1}^i$ are the centers of the new blocks in i th resulting pair. As a result, new clusters $\mathbf{C}_r = \{b_{r1}\}$ and $\mathbf{C}_s = \{b_{s1}\}$, defined by equation (8), take the place of $\mathbf{C}_l$. Based on orthogonal to axis i^* division, the parameters of b_{r1} and b_{s1} are determined as follows:

$$\begin{aligned} \mathbf{L}_{r1}(i) &= \mathbf{L}_{s1}(i) = \mathbf{L}_{l1}(i) \quad i \neq i^* \\ \mathbf{L}_{r1}(i^*) &= \mathbf{L}_{s1}(i^*) = \mathbf{L}_{l1}(i^*)/2 \\ \mathbf{c}_{r1}(i) &= \mathbf{c}_{s1}(i) = \mathbf{c}_{l1}(i) \quad i \neq i^* \\ \mathbf{c}_{r1}(i^*) &= \mathbf{c}_{l1}(i^*) + \mathbf{L}_{l1}(i^*)/4 \\ \mathbf{c}_{s1}(i^*) &= \mathbf{c}_{l1}(i^*) - \mathbf{L}_{l1}(i^*)/4 \end{aligned} \quad (9)$$

This split operation is shown in Fig. 3. As it is illustrated,

a cluster including one constructor block divides at all possible axis-orthogonal directions and a pair of hyper-rectangles, which is nearest to $\mathbf{x}_l$ is selected.

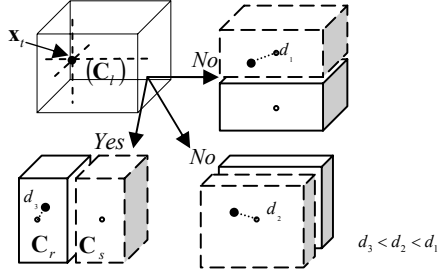


Fig. 3. A scheme of split operation while there is a constructor block in the selected cluster. A pair of clusters is selected, which are closest to $\mathbf{x}_l$.

b) The cluster $\mathbf{C}_l$ of the l th rule consists of more than one constructor block ($N_l > 1, \mathbf{C}_l = \{b_{lj}\}_{j=1}^{N_l}$)

In this case, one of the constructor blocks of $\mathbf{C}_l$, which includes $\mathbf{x}_l$, is removed from $\mathbf{C}_l$ and is considered as a new cluster $\mathbf{C}_r = \{b_{r1}\}$ with one block. The rest of blocks in $\mathbf{C}_l$ are considered as a new cluster, $\mathbf{C}_s$. Corresponding with the two newly created $\mathbf{C}_r$ and $\mathbf{C}_s$, two new rules (r, s) are defined and take the place of rule (l). The linear parameters of new rules ($\mathbf{w}_r^t$ and $\mathbf{w}_s^t$) are initialized equal to $\mathbf{w}_l^t$ as linear parameters of former rule (l). To define $\mathbf{P}_r^t$ and $\mathbf{P}_s^t$ as data matrices of new rules, we can initialize them to be equal to $\mathbf{P}_l^t$ or make a covariance resetting:

$$\mathbf{P}_r^t = \mathbf{P}_s^t = \lambda \mathbf{I}_{(n+1) \times (n+1)} \quad (10)$$

where $\mathbf{I}$ denotes the unit matrix and λ is a large positive constant which is used as the resetting factor. By keeping $\mathbf{P}_j^t; j = r, s$ equal to $\mathbf{P}_l^t$, one forces a slow adaptation of parameters; while covariance resetting will allow for fast tracking of rapid changes.

C.3. Merge procedure

The Merging operation has two main stages: at the first stage, the l th rule is inter-merged with the nearest rule, and then the constructor blocks of this new combined rule are intra-merged together.

a) Inter-merge

In an inter merge, the l th rule is merged with the nearest rule, which is determined as follows:

$$k = \arg \min_j (\|\bar{\mathbf{c}}_l - \bar{\mathbf{c}}_j\|), j \in \{1, 2, \dots, M\} \& j \neq l \quad (11)$$

$$\bar{\mathbf{c}}_w = \frac{\sum_{x=1}^{N_w} V_{wx} \mathbf{c}_{wx}}{\sum_{x=1}^{N_w} V_{wx}}, V_{wx} = \prod_{p=1}^n L_{wx}(p), w \in \{l, j\}$$

where $\bar{\mathbf{c}}_w$ denotes the center of mass of the cluster $\mathbf{C}_w$ and V_{wx} denotes the volume of block $b_{wx} \in \mathbf{C}_w$. Through the inter-merge procedure, the new rule r takes the place of rules l and k , and its cluster, the new cluster $\mathbf{C}_r$, is defined as the union of $\mathbf{C}_l$ and $\mathbf{C}_k$. The linear parameters of rule r , $\mathbf{w}_r$, is determined as a simple interpolation of its building blocks:

$$\mathbf{w}_r = (V_l \mathbf{w}_l + V_k \mathbf{w}_k) / (V_l + V_k) \quad (12)$$

$$V_l = \sum_{x=1}^{N_l} V_{lx}, V_k = \sum_{x=1}^{N_k} V_{kx}$$

One of matrices $\mathbf{P}_l$ and $\mathbf{P}_k$, which has the largest singular value, is chosen as the new data matrix ($\mathbf{P}_r$) to retain the larger rate of adaptation for the new rule. The merge procedure is shown in Fig. 4 where two clusters indicated by different colors are merged together and a new cluster is created.

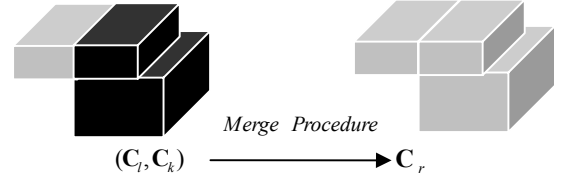


Fig. 4. A scheme of inter-merge operation where two initial clusters with different colors are merged and a new cluster including their constructor blocks is created.

b) Intra-merge

To reduce the complexity of the model and provide a stable learning procedure, it is desirable to decrease the number of internal blocks in each cluster. To this end, every two neighbor hyper-rectangular blocks, which have a common facet, are merged together and a new larger block is made. This type of merging is a simplified version of a former one introduced in [13]. Fig. 5 illustrates this type of merging operation. As it is shown, three constructor blocks are intra-merged together in two steps and a single constructor block is made.

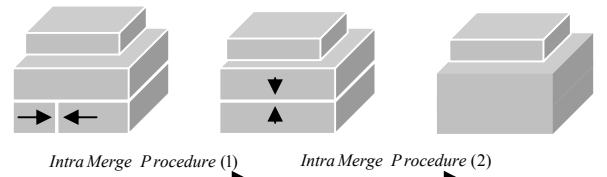


Fig. 5. A simple example that shows intra merging between constructor blocks of a cluster. At each stage, two constructor blocks are merged and the number of constructor blocks is reduced.

C.4. Adaptive threshold

To fulfill the main idea of AHLTNM, adaptive threshold E_t , should be determined appropriately. If E_t is considered too high, no split operation is done and the model always remains an adaptive linear model. In contrast, if E_t is considered too low, the split operations are done continuously and an unfeasible model with a lot of rules is

the result. To prevent this infeasibility of model, $M_{\max}$ is defined as the maximum number of allowed rules in AHLTNM. With respect to $M_{\max}$, E_t is determined as follows:

$$E_t = E_{t-1} + e_0(1 - \text{sgn}(M_{\max} - M)) \quad M \leq M_{\max} \quad (13)$$

where e_0 is computed as $\frac{1}{10}$ of the mean value of absolute errors during $N_0 = 20n$ initial observed data points. As it can be seen in (13), when $M = M_{\max}$, E_t increases gradually with a rate of e_0 ; but when $M < M_{\max}$, E_t does not change. We opt $M_{\max} = 10$ to be used as the default value.

D. The AHLTNM learning Algorithm

Before performing the AHLTNM learning algorithm, an initial hyper-rectangle must be determined and e_0 must be computed through N_0 initial incoming data points. The first hyper-rectangle is determined as large as to include all initial data points as follows:

$$\mathbf{c}_{11} = \frac{\sum_{i=1}^{N_0} \mathbf{x}_i}{N_0}, \quad \mathbf{L}_{11}(k) = u_{\max}^k - u_{\min}^k, \quad k = 1, 2, \dots, n \quad (14)$$

$$u_{\max}^k = \max(\{\mathbf{x}_i(k)\}_{i=1}^{N_0}), \quad u_{\min}^k = \min(\{\mathbf{x}_i(k)\}_{i=1}^{N_0})$$

It is also assumed that the output of the model can be observed after $\tau \geq 1$ due to the input-output delay of the system or considering a multi-step-ahead prediction strategy. The overall algorithm is stated in Table I.

Table I
The overall algorithm of AHLTNM

1-Define parameters of first rule at $t=t_0$: $\mathbf{w}_1^{t_0} = \mathbf{z}_{(n+1) \times 1}, \quad \mathbf{P}_1^{t_0} = \lambda \mathbf{I}_{(n+1) \times (n+1)}$ where $\mathbf{z}$ and $\mathbf{I}$ denote a zero vector and an unit matrix respectively and λ is a large positive constant which is used as the resetting factor. In this paper, $\lambda = 100$ is considered in case studies.
2- For initial $N_0 = 20n$ intake data points, $t := 1$ up to $t = N_0$ perform following tasks 2-1. Estimate the outputs of linear model and absolute errors. 2-2. Update the parameters through the RLS technique. 2-3. Compute e_0 as $\frac{1}{10}$ of the mean value of absolute errors.
3-Define the center and lengths of the first hyper-rectangle using the considered initial N_0 data points through (14); also, initialize adaptive threshold as $E_t = e_0$.
4-Let $t:=t+1$ and compute the output of the model through (2).
5- Update the parameters of the rule (l th one) with highest

validation function at $\mathbf{x}_t$ by using equations (5)-(7).

6- Update the adaptive error threshold, E_t , through (13).

7-If $|e_t| \geq E_t$ and the number of rules is smaller than $M_{\max}$ then run split operation for l th rule by using equations(8)-(10);

8- If $|e_t| < E_t$ and there are at least two rules, run merge operation for l th rule by using equations (11)-(12).

9- Go to step 4.

III. A CASE STUDY

Here, prediction of the monthly smoothed sunspot number time series (as a well studied index of space weather phenomena) is considered. The monthly smoothed sunspot number time series is courtesy of the SIDC (World Data Center for sunspot Index) [14]. In order to compare the prediction results with some of previous studies, all constraints of prediction held before, are considered for AHLTNM too. The training dataset includes samples from 1900 until January 1999 and test dataset includes the part of 23rd solar cycle from January 1999 to May 2001 as explained in [15]. The normalized MSE index (NMSE) is used as the ratio of the sum of square output errors over the sum of square errors between outputs and the averaged values. One-step-ahead prediction of time series ($\tau = 1$) is considered, where y_t is predicted based on $\mathbf{x}_t$ is defined as follows ([15]):

$$\mathbf{x}_t = [y_{t-1}, y_{t-2}, y_{t-3}, y_{t-4}, y_{t-5}]^T \quad (15)$$

The AHLTNM learning algorithm is applied to this considered prediction problem. After $N_0 = 100$ initial obtained data points, $e_0 = 0.0731$ is computed and then an adaptive threshold is initialized and updated through (13). Diagrams of the adaptive threshold and the absolute predicted error, and also, number of rules are shown in Fig. 6.

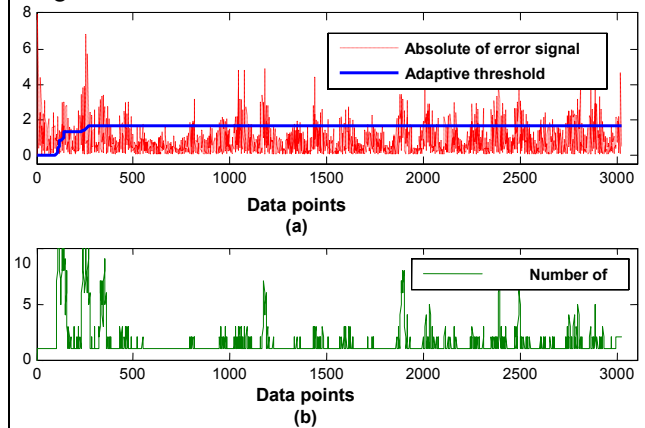


Fig. 6. Plots of adaptive threshold and absolute error, (a), as well as the number of rules, (b), versus the number of obtained data points

As appears in Fig. 6(a), while an absolute error sample (dashed line) is higher than the adaptive threshold (bold

line) a transient nonlinear model is started and is evolved by split operations. As appears in Fig. 6(b), the maximum number of rules in the evolved model is limited to $M_{\max} = 10$. The online model habitually tends to decrease the number of LLMs and to be a linear model. Fig. 7 shows the original and predicted monthly sunspot number time series for the test dataset.

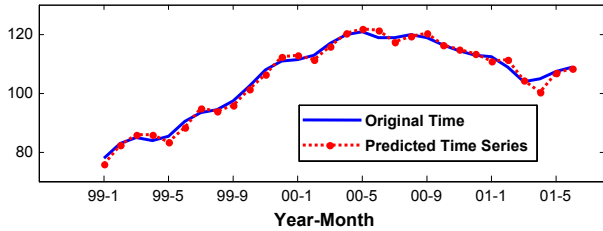


Fig. 7. A graph of the original time series (solid line) and predicted sunspot number time series (dashed-dot line) versus time (year-month)

Table II summarizes the prediction results of AHLTNM learning algorithm and two other online modeling approaches. In all approaches, changing structures for models and parameters is allowed, even through test data. As it can be seen, the AHLTNM has lower NMSE, lower number of rules and a lower execution time compared to the models using from other approaches.

Table II

Global performance measures (NMSE) for recursive prediction of the monthly smoothed sunspot number on January 1999- May 2001

Methods	NMSE	N. fuzzy rules	Execution time
DENFIS [3]	0.0230	11	12 sec
ETS [5]	0.0303	7	15 sec
AHLTNM	0.018	2	7sec

V. CONCLUSION

In this paper, we introduced a predictor model, which can follow the variations of processes with significant uncertainties. The proposed model was a habitually adaptive linear model which can evolve into a TS nonlinear model upon necessity. The necessity basically comes from higher temporal modeling error. Starting with an adaptive linear model helped us to keep the model as simple as possible. A simple model with a few parameters to adapt means more agility in following the variations of the process.

When the linear adaptive model was unable to follow process properly, the model was allowed to evolve into a nonlinear TS model, gradually. The nonlinear model was not a permanent one, however. The proposed AHLTNM learning algorithm is designed with a tendency to return to the linear model. Evolving from a linear model to a nonlinear one and then the return to a linear model were respectively performed through specially designed split and merge procedures. To obtain fast and yet flexible split and merge operations, a modified structure for TS model was considered, which allowed for more complex precedents. An adaptive threshold on temporal modeling error is utilized in the reconstruction of the model. A case study was

considered to examine the applicability and performance of proposed Adaptive Habitually Linear and Transiently Nonlinear Model (AHLTNM) along with its learning algorithm.

The AHLTNM was utilized in prediction of monthly sunspots number. The performance of the proposed model and learning algorithm was compared with two other online modeling approaches. The AHLTNM showed reasonably better performance when faced with significant time-varying behaviors.

REFERENCES

- [1] N. Kasabov, "The ECOS framework and the ECO learning method for evolving connectionist systems," *J. Adv. Comput. Intell.*, vol. 2, no. 6, 1998, pp. 195–202.
- [2] P. Angelov, R. Buswell, "Identification of evolving fuzzy rule based models," *IEEE Trans. on Fuzzy Syst.*, vol. 10, no. 5, 2002, pp. 667–677.
- [3] N. Kasabov, "DENFIS: Dynamic Evolving Neural-Fuzzy Inference System and its application for time-series prediction," *IEEE Trans. Fuzzy Syst.*, vol. 10, 2002, pp. 144–154.
- [4] M.J. Er and S.Q. Wu, "A fast learning algorithm for parsimonious fuzzy neural systems," *Fuzzy Sets and Syst.*, vol. 126, 2002, pp. 337–351.
- [5] P. Angelov, P. Filev, "An approach to online identification of Takagi-Sugeno fuzzy models," *IEEE Trans. Syst., Man, and Cybern. B*, vol. 34, 2004, pp. 484–498.
- [6] Y. Gao and M.J. Er, "NARMAX time series model prediction: feedforward and recurrent fuzzy neural network approaches," *Fuzzy Sets and Syst.*, vol. 50, 2005, pp. 331–350.
- [7] C. Petr, "Online learning of neural Takagi-Sugeno fuzzy model," in *Fuzzy Information Processing Society 2005. NAFIPS 2005. Annual Meeting of the North American*, Jun. 2005, pp. 478–483.
- [8] P. Angelov, X.-W. Zhou, "Evolving fuzzy systems from data streams in real-time," in *Internat. Symp. on Evolving Fuzzy Systems*, 2006, pp. 29–35.
- [9] L. Liao and S. Li, "On-line T-S Fuzzy model identification with growing and pruning rules," in *Proc. of 4th International Symposium on Neural Networks 2007*, Nanjing China, Jun. 2007, pp. 505–511.
- [10] K.M. Pekpe, S. Lecoeuche, "Online, clustering of switching models based on a subspace framework," *Nonlinear Analysis: Hybrid Systems*, vol. 2, no. 3, 2008, pp. 735–749.
- [11] A. Kalhor, B. N. Araabi and C. Lucas, "Online identification of a neuro-fuzzy model through indirect fuzzy clustering of data space," *FUZZ-IEEE2009, the 18th Int. conference on fuzzy systems*, 21–24 Aug., 2009, Korea, pp. 356–359.
- [12] Oliver Nelles, *Nonlinear System Identification*, New York: Springer, 2001, ch.13.3.1.
- [13] A. Kalhor, B. Nadjar Araabi and C. Lucas, "A New Split and Merge Algorithm for Structure Identification in Takagi-Sugeno Fuzzy Model," *7th IEEE International Conference on Intelligent Systems, Design and Applications (ISDA'07)*, Rio de Janeiro, Brazil, October 22–24, 2007, pp. 258–261.
- [14] Solar Influences Data Analysis Center (SIDC), Solar physics research department of the Royal Observatory of Belgium, <http://sidc.oma.be/index.php3>.
- [15] A. Gholipour, B. N. Araabi and C. Lucas, "Predicting chaotic time series using neural and neuro-fuzzy models: A Comparison Study," *Neural Processing Letters*, vol. 24, no. 3, 2006, pp. 217–239.

A Structure Evolved Learning Method for Mamdani Fuzzy Systems

Di Wang, Xiao-Jun Zeng, and John A. Keane

Abstract—This paper proposes a Structure Evolved Learning Method (SELM) for Mamdani Fuzzy Systems. The mechanism of the algorithm is that it is an error-reducing driven learning method. Beginning with a simplest fuzzy system, the algorithm develops the system structure by adding more fuzzy terms and rules to reduce the model errors in a ‘greedy’ way. The main features of the proposed algorithm are that, firstly, it can automatically determines and controls the number and location of fuzzy terms needed by following the error-reducing driven evolving process to achieve the desired accuracy; secondly, it adds new fuzzy terms and rules by evenly distributing error to each sub-region aiming at an efficient set of fuzzy rules, thirdly, it uses triangular membership functions with the regular partitions and Mamdani fuzzy systems and leads to identified fuzzy system models with good transparency and interpretability. Two benchmark examples are given to illustrate the advantages of the proposed algorithm.

Keywords- fuzzy systems, evolved learning, fuzzy systems identification.

I. INTRODUCTION

Since the invention of fuzzy sets and fuzzy systems by Zadeh four decades ago, fuzzy systems have been successfully applied in different areas. There are several fundamental reasons for the achievements of fuzzy systems [1]-[7]: firstly, fuzzy systems can be identified by combining human knowledge and learning from data; secondly, the transparency and interpretability of fuzzy rules and systems make them easy to understand, use and judge in applications, thirdly, fuzzy systems are universal approximators which can approximate any continuous nonlinear function to any degree of accuracy and therefore are applicable to be used to solve general complex and nonlinear problems.

In traditional fuzzy systems, fuzzy terms are defined by a global grid-based partition. There are three drawbacks by this global grid-based definition. Firstly, a global grid-based partition inherently suffers the exponential growth of fuzzy rules and parameters corresponding to the number of input variables; Secondly not all the fuzzy terms obtained by a global grid-based partition on each input variable are effective; Thirdly, this fixed definition of global grid-based

partition makes the whole fuzzy systems unintelligent and inflexible.

To overcome these weaknesses, various different methods have been developed. Adaptive fuzzy or neuro-fuzzy method has become one of the most important approaches in this aspect [8]-[37]. In particular, eTS, DENFIS, SOFNN, and SAFIS have been proposed by Angelov[10]-[11], Kasabov [12], Leng and al [13]-[14], and Rung and al [15] respectively. These methods have been proved having the capability to automatically identify the structure and parameters of complex nonlinear systems with good numerical accuracy.

Motivated by the weaknesses in traditional fuzzy systems, this paper proposes a Structure Evolved Learning Method for (SELM) fuzzy systems identification. SELM starts with a simplest fuzzy system with two fuzzy terms for each input variable and develops the structure of the fuzzy system by increasing fuzzy terms and rules on the selected input variable in the selected sub-region step by step. Different from the existing methods such as eTS, DENFIS, SOFNN, and SAFIS, the main features of SELM are as follows:

- SELM is an error-reducing driven method. That is, the structure of a fuzzy system evolves along with the aim of reducing the errors until the model error is within an accepted threshold. In other words, the method is focusing directly the goal of learning – to represent a given system with a desired accuracy. This leads to the mechanism of SELM in structure identification, that is, SELM automatically determines the number and location of fuzzy terms needed by following the error-reducing driven evolving process to achieve the desired accuracy.
- SELM is a greedy incremental process with the goal to achieve the most effective error-reducing. For this purpose, fuzzy terms and rules are added at point within selected region which evenly divides its local error along the input variable to get the best accuracy in one step. This step-by-step adding can be regarded as ‘greedy construction’ strategy. As all ‘greedy’ methods, SELM makes the locally optimum choice at each step of incremental construction with the hope of finding the global optimum [18]-[19].
- SELM uses triangular membership functions with regular partitions and Mamdani fuzzy systems in order to achieve the transparency and interpretability of the resulting fuzzy systems.

Di Wang is with Manchester Business School, the University of Manchester, Booth Street West, Manchester, M15 6PB (e-mail: wangdi_2001@hotmail.com).

Xiao-Jun Zeng and John A. Keane are with the School of Computer Science, the University of Manchester, Manchester, M13 9 PL, UK (e-mail: x.zeng@manchester.ac.uk, John.Keane@manchester.ac.uk).

This paper is organized as follows: Section II presents Structure Evolved Learning Method (SELM) for Fuzzy Systems. Section III presents the proposed SELM algorithm for Mamdani fuzzy system. Section IV gives two examples to illustrate the advantages of the proposed algorithm. Finally Section V summarizes the main conclusion.

II. STRUCTURE EVOLVED LEARNING METHOD FOR FUZZY SYSTEMS

A. Structure evolution

In this section, we introduce the proposed scheme for evolved structure learning for fuzzy systems. We are using triangular membership functions with regular partitions and Mamdani fuzzy systems to achieve the better transparency and interpretability.

Initially we consider the problem definition domain as a whole region. We then split this whole region into two sub-regions to obtain even error distribution between these two sub-regions. Then we select one sub-region with maximal average error for further splitting. This splitting process continues until we obtain the satisfactory accuracy. In this incremental construction process, a sub-region having the maximal average local error is always selected for further splitting.

For the selected sub-region, we locate a proper data instance candidate along each attribute (the i th attribute, i.e. the input variable x_i) which evenly split the accumulated error for the selected sub-region. Then one attribute $x_s^{splitting}$ is selected for splitting if the best global accuracy is obtained comparing with splitting on other attributes.

The splitting point $x_s^{splitting}$ is identified as a new partition point on that input variable. The main idea behind this can be explained as follows: We partition the input variables unevenly step by step. The aim of this uneven partition is to distribute error more evenly to each sub-region in next step. This even error distribution policy helps to create the most efficient new fuzzy terms and rules.

We can visualize this uneven splitting process by Fig.1. Fig. 1 (a) shows the definition domain and Fig. 1 (b) shows the case to construct a fuzzy system with a global even grid-based partition. This even grid-based partition is predefined. Fig. 1 (c) and Fig. 1 (d) show the incremental splitting for SELM. If in Fig. 1 (c) the average error in Sub-region 1 is larger than that in Sub-region 2, then Sub-region 1 is selected for further splitting. It is split into Sub-region 1' and Sub-region 3 in Fig. 1 (d). Comparison with global even grid-based partition, we see that the proposed incremental construction scheme avoids the exponentially increasing of fuzzy terms/rules suffered by the global even grid-based partition as the number of sub-regions increases by only one each time.

Remark1: SELM is error-reducing driven and aims at reducing the errors by adding rules step by step. As other methods for system identification, SELM needs to balance

accuracy and system complexity. Too few rules result in under fitting and inaccurate results whilst too many rules result in complicated system structure and over fitting (especially in presence of noise). To avoid over fitting, we need to set a proper expected accuracy as a stop critical for system splitting. A proper splitting should result in a system with similar training accuracy and testing accuracy.

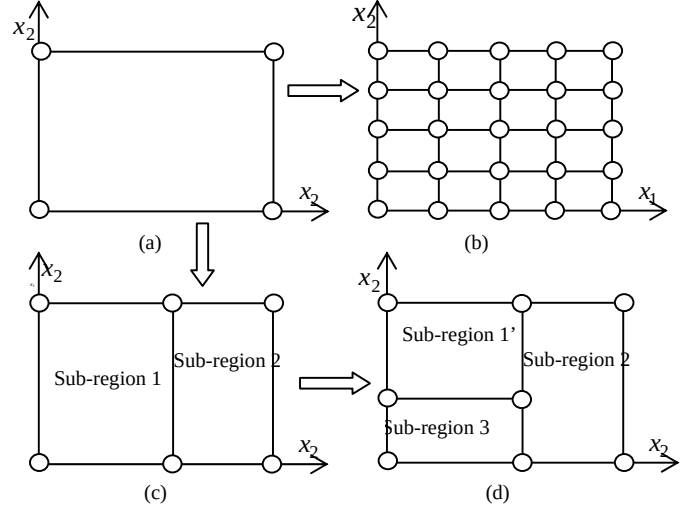


Fig.1. Sub-region creation for traditional fuzzy system and SELM

B. Formulations

In this section, to explain the formula for SELM, we consider a problem with n input variables.

As described in Sub-section II.A, SELM starts with one whole region. The definition domain for an n -dimensional problem is $\left(\times_{i=1}^n [b_i, c_i] \right)$, where b_i and c_i are the minimal and

maximal value for the i th attribute. Then the initial Mamdani fuzzy system has the rule base as follows:

IF x_1 is $A_{1,1}^{k_1}$, and x_2 is $A_{1,2}^{k_2}$, ... and x_n is $A_{1,n}^{k_n}$, THEN y is $y^{k_1 k_2 \dots k_n}$, (1)

where k_i is 0 or 1. If $k_i = 0$, then $A_{1,i}^{k_i} = A_{1,i}^0 = b_i$; if $k_i = 1$, then $A_{1,i}^{k_i} = A_{1,i}^1 = c_i$. $y^{k_1 k_2 \dots k_n}$ is parameters to be identified.

The fuzzy system starts from the simplest rule base (1), and continues with sub-region splitting process on the selected domain. In the following discussion, m is used to indicate the index of the selected sub-region for further splitting. The sub-region having the maximal average local error is always selected for further splitting.

Assume that input vector $\mathbf{x}$ belongs to the m th sub-region, i.e., $\mathbf{x} = (x_1, x_2, \dots, x_n) \in \times_{i=1}^n [b_{m,i}, c_{m,i}]$, where $b_{m,i}$ and $c_{m,i}$ are the minimal and maximal value for the i th attribute x_i for the m th sub-region. Then the fuzzy rule base for the m th sub-region is:

IF x_1 is $A_{m,1}^{k_1}$, and x_2 is $A_{m,2}^{k_2}$, ... and x_n is $A_{m,n}^{k_n}$, THEN y is $y_m^{k_1 k_2 \dots k_n}$, (2)

where k_i is 0 or 1. If $k_i = 0$, then $A_{m,i}^{k_i} = A_{m,i}^0 = b_{m,i}$; if $k_i = 1$, then $A_{m,i}^{k_i} = A_{m,i}^1 = c_{m,i}$,

The regular triangular membership functions for that the m th sub-region (in Equation (2)) are defined as follow.

The membership function for fuzzy term $A_{m,i}^0(x_i)$ (when $k_i = 0$) is:

$$\mu_{m,i}^0(x_i) = \frac{c_{m,i} - x_i}{c_{m,i} - b_{m,i}}. \quad (3)$$

The membership function for fuzzy term $A_{m,i}^1(x_{m,i})$ (when $k_i = 1$) is:

$$\mu_{m,i}^1(x_i) = \frac{x_i - b_{m,i}}{c_{m,i} - b_{m,i}}. \quad (4)$$

Then the output for the m th sub-region is:

$$O_m = \frac{\sum_{k_1 k_2 \dots k_n} (A_m^{k_1 k_2 \dots k_n}(\mathbf{x}) \times y_m^{k_1 k_2 \dots k_n})}{\sum_{k_1 k_2 \dots k_n} A_m^{k_1 k_2 \dots k_n}(\mathbf{x}) |_{\mathbf{x} \in \text{domain}_m}} \quad (5)$$

where

$$\sum_{k_1 k_2 \dots k_n} A_m^{k_1 k_2 \dots k_n}(\mathbf{x}) |_{\mathbf{x} \in \text{domain}_m} = 1 \quad (6)$$

$$A_m^{k_1 k_2 \dots k_n}(\mathbf{x}) = \prod_{i=1}^n (\mu_{m,i}^{k_i}(x_i)). \quad (7)$$

Based on (6), Equation (5) can be written as:

$$O_m = \sum_{k_1 k_2 \dots k_n} A_m^{k_1 k_2 \dots k_n}(\mathbf{x}) \times y_m^{k_1 k_2 \dots k_n} |_{\mathbf{x} \in \text{domain}_m} \quad (8)$$

Through the definition of regular triangle membership function (3), (4) and (7), an input is uniquely located in one sub-region and activates only the membership functions at the edge of that sub-region. That is, the overall fuzzy system

$$o(X) = o_m(X) \quad X \in \text{domain}_m \quad (9)$$

We can see from (8) the model output of an input vector $\mathbf{x}$ within the m th sub-region o_m is determined by both the consequent parameters $y_m^{k_1 k_2 \dots k_n}$ and the membership functions $A_m^{k_1 k_2 \dots k_n}(\mathbf{x})$ of the m th sub-region in which they are located.

C. Fuzzy term/rule creation and membership function updating

In each sub-region splitting step, one new fuzzy term

(membership function) on the selected splitting attribute is created and the two membership functions on the selected splitting attribute on the edge of the old selected sub-region are updated. We now explain how to create new fuzzy terms and rules and how to update related membership functions.

For a selected sub-region (with index of m), we assume s is the index for the selected attribute for splitting. This attribute selection criterion is described in subsection II.A.

If we represent the splitting point by $x_s^{\text{splitting}}$, then one new fuzzy term $x_s^{\text{splitting}}$ is added and its associated membership function is created as follows:

$$\mu_s^{\text{splitting}}(x_s) = \begin{cases} \frac{x_s - b_{m,s}}{x_s^{\text{splitting}} - a_{m,s}}, & \text{if } x_s \in [b_{m,s}, x_s^{\text{splitting}}) \\ \frac{c_{m,s} - x_s}{c_{m,s} - x_s^{\text{splitting}}}, & \text{if } x_s \in [x_s^{\text{splitting}}, c_{m,s}) \end{cases} \quad (10)$$

After creating a new membership function, the two membership functions on selected splitting attribute x_s on the edge of the (old) selected (the m th) sub-region (represented by Equation (3) and (4)) are updated by (11) and (12).

$$\mu_{m,s}^0(x_s) = \frac{x_s^{\text{splitting}} - x_s}{x_s^{\text{splitting}} - b_{m,s}}. \quad (11)$$

$$\mu_{m,s}^1(x_s) = \frac{x_s - x_s^{\text{splitting}}}{c_{m,s} - x_s^{\text{splitting}}}. \quad (12)$$

Along with the creation of one fuzzy term (membership function), 2^{n-1} new fuzzy rules are created as:

IF x_1 is $A_{m,1}^{k_1}$, and x_2 is $A_{m,2}^{k_2}$, ..., and x_i is $x_i^{\text{splitting}}$, ..., and x_n is $A_{m,n}^{k_n}$, THEN Y is". $y^{k_1 \dots k_{s-1} k_{s+1} \dots k_n}$ (13)

where k_i is 0 or 1 and $i \neq s$.

We now visualize this new fuzzy term (membership function)/rule creation process and the membership function updating process by a two-dimensional example in Fig.2: The white points are fuzzy rules; the black points are data points for splitting. $b_{m,i}$ and $c_{m,i}$ are the minimal and maximal value for the i th attribute for the m th sub-region. In Fig. 2 (a), assuming sub-region₂ is the selected sub-region with maximal average local error, $x_1^{\text{splitting}}$ is the selected point for splitting, where $\mathbf{x}^{\text{splitting}} = (x_1^{\text{splitting}}, x_2^{\text{splitting}})$ is a data instance in the dataset. Sub-region₂ is then split into Sub-region'₂ and Sub-region₃ by $x_1^{\text{splitting}}$. Two new fuzzy rules are created at follows as shown in Fig. 2 (b).

$$\text{IF } x_1 \text{ is } x_1^{\text{splitting}}, x_2 \text{ is } b_{2,2}, \text{ THEN } Y \text{ is } y_{\text{splitting}}^0 \quad (14)$$

$$\text{IF } x_1 \text{ is } x_1^{\text{splitting}}, x_2 \text{ is } c_{2,2}, \text{ THEN } Y \text{ is } y_{\text{splitting}}^1 \quad (15)$$

The membership functions for Sub-region₂ are as shown as

in Fig. 2 (c) (before splitting) and Fig. 2 (d) (after splitting). In fig. 2 (d), the middle triangle is the membership function for the new created fuzzy term. The two half triangles on the edges are membership functions after updating.

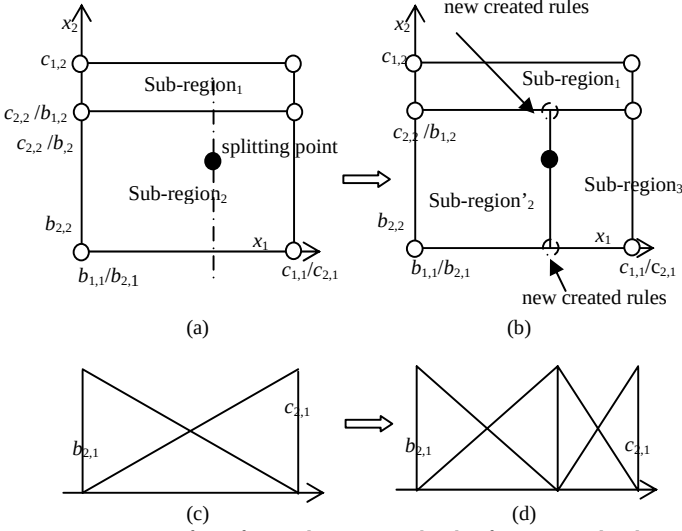


Fig.2. Creation of new fuzzy rules, new membership functions, and updating of existing fuzzy rules and membership functions.

D. Parameter initialization and optimization for new created fuzzy rules

The initial values of newly created parameters $y_{k_1 k_2 \dots k_{s-1} k_{s+1} \dots k_n}$ are set as follows.

$$y_{k_1 k_2 \dots k_{s-1} k_{s+1} \dots k_n} = \sum_{k'_1 k'_2 \dots k'_n} \left(\prod_{i=1}^n \mu_{m,i}^{k'_i}(x_i) \right) \times y_m^{k'_1 k'_2 \dots k'_n} \quad (16)$$

where $x_i = b_{m,i}$, when $k_i = 0$; $x_i = c_{m,i}$, when $k_i = 1$, and $x_s = x_s^{splitting}$.

From equation (1) and (2), we have: $\mu_{m,i}^{k'_i}(x_i) = 0$, when $k_i \neq k'_i$; $\mu_{m,i}^{k'_i}(x_i) = 1$, when $k_i = k'_i$. If $b_s^{splitting} \leq x_s^{splitting} \leq c_s^{splitting}$, when

$$i = s : \mu_{m,s}^{k'_s}(x_s) = \frac{c_{m,s} - x_s^{splitting}}{c_{m,s} - b_{m,s}} \quad \text{if } k_s = k'_s = 0 ;$$

$$\mu_{m,s}^{k'_s}(x_s) = \frac{x_s^{splitting} - c_{m,s}}{b_{m,s} - c_{m,s}} \quad \text{if } k_s = k'_s = 1 .$$

Based on (16), this implies

$$y_{k_1 k_2 \dots k_{s-1} k_{s+1} \dots k_n} = \left(\frac{c_{m,s} - x_s^{splitting}}{c_{m,s} - b_{m,s}} \right) \times y_m^{k_1 k_2 \dots k_n} \Big|_{k_s=0} + \left(\frac{x_s^{splitting} - b_{m,s}}{c_{m,s} - b_{m,s}} \right) \times y_m^{k_1 k_2 \dots k_n} \Big|_{k_s=1} \quad (17)$$

By doing this, we obtain a good starting point for the following training.

III. ALGORITHM

We now generalize the algorithm for SELM.

1. The fuzzy system starts from one (sub-) region, the definition domain $\left(\times_{i=1}^n [b_i, c_i] \right)$. The fuzzy terms and rule base are initialized by the minimal value b_i and maximal value c_i ($i = 1 \dots n$) for each attribute. The detailed formulas for these initial fuzzy rules are described as (1)
2. Compute the average local error, $error_j^{average}$ for each sub-region j and select one sub-region: the m th sub-region with maximal average local error: $m = \arg \min_{j=1}^M (error_j^{average})$, where M is the total number of current sub-regions, and $error_j^{average}$ is defined as:

$$error_j^{average} = \left(\frac{1}{N_j} \times \sum_{x^k \in \text{Sub-region}_j} (o^k - y^k)^2 \right)^{\frac{1}{2}}, \quad (18)$$

assuming that N_j data instances locate in the j th sub-region, o^k and y^k are the model output and actual output of x^k .

3. Within the selected (the m th) sub-region, along each attribute i ($i = 1 \dots n$) we find a data instance with $x_i^{splitting}$ which evenly separates the accumulated error for Sub-region m . Try to split Sub-region m into two sub-regions at each $x_i^{splitting}$ along the i th attribute ($i = 1 \dots n$). The attribute $x_s^{splitting}$ with index s is selected if splitting sub-region m at $x_s^{splitting}$ results in the best global accuracy: $s = \arg \min_{i=1}^n (error_i^{global})$, where $error_i^{global}$ is the global error after splitting Sub-region m at $x_i^{splitting}$.
4. Split Sub-region m into two sub-regions and increase the total number of sub-regions by one: $M = M + 1$.
5. Based on the splitting in Step 4, create new fuzzy terms according to Equation (10), update the existing fuzzy terms according to Equation (11) and (12) and create new fuzzy rules as (13). The values for parameters in the new created fuzzy rules are initialized using Equation (17).
6. Re-training parameters for global optimization after splitting.
7. If M is larger than the maximal limit of sub-regions or the global accuracy is better than the requirement,

then stop. Else continue the evolving process with Step 2.

Remark2: By using the above process to create new fuzzy terms and rules, the maximal number of fuzzy rules is $2^n + (MaxReg - 1) \times 2^{n-1} = (MaxReg + 1) \times 2^{n-1}$, and the maximal total number of fuzzy terms is $2^n + (MaxReg - 1)$, where $MaxReg$ is the maximal number of sub-regions and n is the input dimension. With such a maximal allowed number of sub-regions $MaxReg$, the overall complexity of the final fuzzy model and the number of fuzzy terms allowed are controlled.

IV. SIMULATION

In this section, two benchmark system identification examples are given to illustrate the advantages of the proposed algorithm.

Simulation 1: Consider the following widely used two-dimensional non-linear function approximation problem [1], [26]-[28];

$$y = f(x_1, x_2) = (1 + x_1^{-2} + x_2^{-1.5})^2 \quad x_1 \in [1, 5], \quad x_2 \in [1, 5].$$

In this simulation, 50 training samples are randomly generated on the definition domain for both training and test. The performance comparison by different methods in MSE is reported in Table 1. The number of parameters in the third column is the number of parameters used in the consequent part. We can see from Table 1 that SELM obtains better accuracy than other training algorithms when using the same number of parameters.

TABLE 1. PERFORMANCE COMPARISON IN MSE WITH OTHER ALGORITHMS FOR SIMULATION 1

Method	Number of rules	Number of parameters	MSE
Sugeno and Yasukawa [28]	6	22	0.0790
Emami and Turksen [1]	8	24	0.0040
Tsekouras [27]	8	24	0.0042
Kim et al [26]	3	9	0.0550
SELM	8	8	0.0410
	10	10	0.0018
	12	12	0.0012
	14	14	0.0009

Simulation 2: In this simulation, we consider the non-linear dynamic system identification problem used by SAFIS [15], MRAN [27], RANEKF [28], Simp_eTS [11], eTS [10] and HA[29] as follows:

$$y(t+1) = \frac{y(t) \times y(t-1) \times [y(t) - 0.5]}{1 + y(t)^2 + y(t-1)^2} + u(t)$$

where $u(t) = \sin\left(\frac{2\pi t}{25}\right)$, $y(0) = 0$, and $y(1) = 0$.

5000 data were created for training, and 300 data were created for testing. The performance comparison of the proposed ICLA with SAFIS, MRAN, RANEKF, Simp_eTS, eTS and HA in RMSE is shown in Table 2. From Table 2, we see that the proposed SELM uses fewer fuzzy rules but

achieves better accuracy than all the other methods. In additions, SELM products a more transparent fuzzy system due to using regular triangular membership functions.

TABLE 2 PERFORMANCE COMPARISON IN RSME WITH OTHER ALGORITHMS FOR SIMULATION 1

Method	Number of rules/neurons	RMSE for training	RMSE for testing
SAFIS [17]	17	0.0539	0.0221
MRAN [29]	22	0.0371	0.0271
RANEKF [30]	35	0.0273	0.0297
Simp_eTS [11]	22	0.0528	0.0225
eTS [10]	49	0.0292	0.0212
HA [31]	28	0.0182	0.0244
SELM	8	0.0318	0.0317
SELM	12	0.0039	0.0038
SELM	16	0.0013	6.7663E-4
SELM	20	0.0010	7.79243E-4

V. CONCLUSIONS

This paper proposes a Structure Evolved Learning Method (SELM) for the identification of Fuzzy Systems. Comparing with traditional learning methods of fuzzy systems, the motivation of SELM is to overcome the following main weaknesses:

- The even global grid-based partitions of fuzzy terms leads to the ineffectiveness in the sense that the rules and parameters increases exponentially with the number of input variables;
- The fixed definition of global grid-based partitions leads to the resulting fuzzy systems unintelligent and inflexible.

There are two key ideas behind the proposed SELM:

- The first is that the fuzzy partition of input spaces should follow the different smoothness of the approximated nonlinear function in the different regions. The more sparse (or coarse) partitions for the regions where the approximated function is smoother and the denser (or finer) partitions for the regions where the approximated function is less smooth.
- The second is that the incremental and evolving learning process is to follow the fastest routine to achieve the approximation error reduction.

SELM starts with the simplest fuzzy system, and develops its structure by adding fuzzy terms and rules based on the appropriately selected sub-region, input variable and point for splitting. SELM increases the number of sub-regions and fuzzy rules along the direction of the fastest approximation error reduction until it meets some pre-defined accuracy requirement. It can be proved that, by adding new fuzzy rules, the new fuzzy system can always achieve better accuracy than the old one. The reason is that we make use of a ‘greedy’ approach to find the sub-region, the splitting point and the attribute to create new fuzzy rules to obtain the least error in the next step by sub-region splitting. To verify the effectiveness and efficiency of SELM, two benchmark simulation examples have been carried out and compared

with the various evolving learning methods such as eTS, DENFIS, and SAFIS. These simulation results show that SELM often obtains better accuracy by using the same or less number of fuzzy rules.

REFERENCES

- [1] M. R. Emami, B. Turksen and A.A. Goldenberg, "Development of a systematic methodology of fuzzy modeling", IEEE Trans. Fuzzy Syst., Vol. 6, No. 6, pp. 341-361, 1998.
- [2] Witold Pedrycz, "Logic-Based Fuzzy Neurocomputing with Unineurons", IEEE Transactions on Fuzzy Systems, Vol. 14, No. 6, pp. 860-873, 2006.
- [3] T. Takagi and M. Sugeno, "Fuzzy Identification of Systems and its Applications to Modeling and Control", IEEE Transactions on Systems, Man and Cybernetics, Vol. 15, No. 1, pp. 116-132, 1985.
- [4] L.-X. Wang, "Fuzzy Systems are Universal Approximators", Proceeding 1st IEEE Conference on Fuzzy Systems, pp. 1163-1169, San Diego, CA, 1992
- [5] L.-X. Wang and J. M. Mendel, "Fuzzy basis functions, universal approximation, and orthogonal least-squares learning," IEEE Trans. Neural Networks, vol. 3, pp. 807-814, 1992.
- [6] X.-J. Zeng and M.G. Singh, "Approximation theory of fuzzy systems---SISO case," IEEE Trans. Fuzzy Systems, Vol. 2, pp. 162-176, 1994.
- [7] X.-J. Zeng and M.G. Singh, "Approximation theory of fuzzy systems---MIMO case," IEEE Trans. Fuzzy Systems, Vol. 3, pp. 219-235, 1995.
- [8] P. Angelov, *Evolving Rule-based Models: A Tool for Design of Flexible Adaptive Systems*, Springer-Verlag, Heidelberg, New York, 2002.
- [9] P. Angelov, D. Filev, N. Kasabov, and O. Cordon (Eds.), *Evolving Fuzzy Systems*, Proc. of the 2006 International Symposium on Evolving Fuzzy Systems EFS'06, IEEE Press, 2006.
- [10] P.P. Angelov and D.P. Filev, "An approach to online identification of Takagi-Sugeno fuzzy models," IEEE Trans. Systems Man Cybernet.—Part B, Vol. 34, pp. 484-498, 2004.
- [11] P. Angelov, D. Filev, "Simpl_eTS: a simplified method for learning evolving Takagi-Sugeno fuzzy models," in: The 14th IEEE Internat. Conf. on Fuzzy Systems, pp. 1068-1073, 2005.
- [12] N. K. Kasabov, "DENFIS: Dynamic Evolving Neural-Fuzzy Inference System and its application for time-series prediction", IEEE Transactions On Fuzzy Systems. Vol. 10, pp. 144-153, 2002.
- [13] G. Leng, T. M. McGinnity and G. Prasad, "An Approach for On-line Extraction of Fuzzy Rules using a Self-organising Fuzzy Neural Network", Fuzzy Sets and Systems, Vol. 150, No. 2, pp. 211-243, 2005.
- [14] G. Leng, G. Prasad, and T. M. McGinnity, (Dec 2004) "An on-line algorithm for creating self-organising fuzzy neural networks", Neural Networks, Vol. 17, No. 10, pp. 1477-1493, 2004.
- [15] H. J. Rung, N. Sundararajan, G.B. Huang and P. Saratchandran, "Sequential Adaptive Fuzzy Inference System (SAFIS) for nonlinear system identification and prediction," Fuzzy System and Set, Vol. 157, pp. 1260-1275, 2006.
- [16] E. Jack., "Matroid and the greedy algorithm", Math. Programming Vol. 1, pp. 127-113, 1971.
- [17] K. Bernard, L. Lázsló, "Mathematical structures underlying greedy algorithms", in Gecseg, F. (Ed.), *Fundamentals of Computation Theory: Proceedings of the 1981 International FCT-Conference*, Lecture Notes in Comput. Sci. Vol. 117, pp. 205-209, Berlin: Springer-Verlag, 1981.
- [18] L.-X. Wang, *A Course in Fuzzy Systems and Control*. Upper Saddle River, NJ: Prentice-Hall, 1997.
- [19] Y. S. Ding, H. Ying, and S. H. Shao, "Necessary Conditions on Minimal System Configuration for General MISO Mamdani Fuzzy Systems as Universal Approximators", IEEE Trans. Syst. Man and Cybern. - B, Vol. 30, No. 6, pp. 857-864, 2000.
- [20] W. Pedrycz, "Linguistic models as a framework of user-centric system modeling", IEEE Trans. Syst. Man and Cybern. - A, Vol. 36, No. 4, pp. 727-745, 2006.
- [21] E. Kim, M. Park, S. Ji and M. Park, "A new approach to fuzzy modeling", IEEE Trans. Fuzzy Syst., Vol. 5, No. 3, pp. 328-337, 1997.
- [22] G. E. Tsekouras, "On the use of the weighted fuzzy C-means in fuzzy modeling", Int. J. Advances in Eng. Software, Vol. 36, pp. 287-300, 2005.
- [23] M. Sugeno and T. Yasukawa, "A fuzzy-logic based approach to qualitative modeling", IEEE Trans. Fuzzy Syst., Vol. 1, No. 1, pp. 7-31, 1993.
- [24] <http://lib.stat.cmu.edu/datasets/>.
- [25] S. Wu and M. J. Er, "Dynamic fuzzy neural networks-a novel approach to function approximation", IEEE Trans. Syst. Man and Cybern. - B, Vol. 30, pp. 358-364, 2000.
- [26] S. Wu, M. J. Er, and Y. Gao, "A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks", IEEE Trans. Fuzzy Syst., Vol. 9, pp. 578-594, 2001.
- [27] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks", Neural Comput. Vol. 9, pp. 461-478, 1997.
- [28] Kadir Kamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks", Neural Comput. Vol. 5, pp. 954-975, 1993.
- [29] L. Wang and J. Yen, "Extracting fuzzy rules for system modeling using a hybrid of genetic algorithm and Kalman filter", Fuzzy Sets and Systems, Vol. 101, pp. 353-362, 1999.
- [30] B. Fritze, "A growing neural gas network learns topologies," Advances in Neural Information Processing Systems, Vol. 7, pp. 1-8, 1995.
- [31] J. Platt, "A resource allocating network for function interpolation," Neural Computation, Vol. 3, pp. 213-225, 1991.
- [32] D. Deng and N. Kasabov, "Evolving self-organizing maps for online learning, data analysis and modeling", in Proceeding IJCNN 2000, Neural Networks Neural Computing, New Changes Perspectives New Millennium, Vol. VI, S. -I. C.L. Giles, M. Gori and V. Piuri, Eds. New York, pp. 3-8, 2000.
- [33] N. Kasabov, "Evolving fuzzy neural networks – algorithms, applications and biological motivation", in Methodologies for conception design and application of soft computing, T. Yamakawa, and G. Matsumoto, Eds, Singapore: World Scientific, pp. 217-274, 1998.
- [34] S.J. Mousavi, K. Ponnambalam and F. Karray, "Inferring operating rules for reservoir operations using fuzzy regression and ANFIS," Fuzzy Systems and Sets, Vol. 158, No. 10, pp. 1064-1062, 2007.
- [35] H. Qin and S. X. Yang, "Adaptive neuro-fuzzy inference systems based approach to nonlinear noise cancellation for images", Fuzzy Systems and Sets, Vol. 158, No. 10, pp. 1036-1063, 2007.
- [36] Faa-Jeng Lin, and Po-Hung Shen, "Adaptive Fuzzy-Neural-Network Control DSP-Based Permanent Magnet Linear Synchronous Motor Servo Drive", IEEE Transactions On Fuzzy Systems, Vol. 14, No. 4, pp. 481-495, 2006.
- [37] H. Ishibuchi, K. Nozaki and H. Tanaka, "Efficient fuzzy partition of pattern space for classification problems", Fuzzy Sets and Systems, Vol. 59, pp. 295-304, 1993.
- [38] Gang Feng, "A Survey on Analysis and Design of Model-Based Fuzzy Control Systems", IEEE Transactions On Fuzzy Systems, Vol. 14, No. 5, pp. 676-697, 2006.

Dynamic Analysis of the Participatory Learning Algorithm

Elton Lima, Rosangela Ballini and Fernando Gomide

Abstract—This paper concerns the analysis of the dynamic behavior of the participatory learning process considering the compatibility and the arousal mechanisms simultaneously with the learning rate. The purpose is to study the convergence characteristics of participatory learning algorithm when learning data are a continuous and differentiable function of the learning variable. The main feature of the participatory learning model is variable compatibility and arousal values that entails variable learning rates. This feature involves the important problem of proving the stability and convergence of the learning system for this condition.

I. INTRODUCTION

Participatory learning is an approach for learning that emphasizes the role of the current knowledge as a part of the learning process itself. Contrary to many learning paradigms, the participatory learning process includes a compatibility mechanism to let current knowledge influence how data affect learning. The impact of data in causing learning and knowledge revision depends on compatibility of data with the current knowledge [1].

One of the main applications of participatory learning is in incremental clustering, an essential element of information processing and machine learning systems [2]. Participatory learning also is part of many instances of data mining, evolving system modeling [3], control, forecasting [4] and classification systems.

Yager suggested in [1] an instance of participatory learning in the form of a basic exponential smoothing like model. The view of participatory learning as a combination of two regulatory mechanisms, participatory feedback mechanism and participatory arousal mechanism, was pursued in [5]. In particular, it was shown that the basic feedback participatory mechanism is stable whenever the compatibility of data with the current knowledge stays within the unit interval. In the present paper we analyse the dynamic behavior of the participatory learning process considering the compatibility and the arousal mechanisms simultaneously with the learning rate. More specifically, we look at the convergence of participatory learning algorithm when learning data are a function of the learning variable.

II. PARTICIPATORY LEARNING

An essential feature of the participatory learning paradigm is that input data has the greatest impact in causing learning and knowledge revision when it is compatible with the current system knowledge. In particular, data too conflicting

with the current knowledge are discounted. Let $v \in [0, 1]^p$ be the object of learning, a variable encoding the knowledge of a system. We may say that our aim is to learn a value for v using data $x^k \in [0, 1]^p$, $k = 1, 2, \dots$. The current knowledge of the system consists of the vector v^k whose components, v_i^k , $i = 1, 2, \dots, p$, are the system current knowledge about the values of the x_i^k . This is what has been learned after $k-1$ data are input.

Using the participatory learning mechanism [1], the current knowledge is updated using the following expression

$$v^{k+1} = v^k + \alpha G^k(x^k - v^k) \quad (1)$$

where $v^k \in [0, 1]^p$, $\alpha \in [0, 1]$ is the basic learning rate, and $G^k = G(\rho(x^k, v^k))$ is a function of the compatibility $\rho(x^k, v^k)$ of data x^k with current knowledge v^k at $k = 1, 2, \dots$.

Compatibility can be viewed as an acceptance mechanism. The larger the $\rho(x^k, v^k)$, the more compatible the data is with the current knowledge. If most of data is compatible with the current knowledge, then we can use the portion of the data that deviates from the current knowledge to learn. A simple choice is $G^k = \rho^k$, that is, G^k is the identity function. This choice gives the basic participatory learning expression

$$v^{k+1} = v^k + \alpha \rho^k(x^k - v^k) \quad (2)$$

where

$$\rho^k = 1 - d(x^k, v^k) \quad (3)$$

and d is a distance measure between x^k and v^k .

Another important mechanism in participatory learning is the arousal mechanism. The arousal mechanism is an autonomous element, independent of the current knowledge, that observes the performance of the compatibility. The arousal mechanism controls compatibility to consider the possibility that the current knowledge may be erroneous. It essentially does this by noting how often incompatible data have been input. Its effect is not manifested by an individual incompatibility, but by an accumulation of a continued stream of low compatibilities. This should make the learning process more susceptible to learning. If too many observations are rejected as being incompatible with the current knowledge, then current knowledge should be revised.

The arousal mechanism can be viewed as a value $a^k \in [0, 1]$ called the arousal rate. It is inversely related to the confidence in the model based upon its past performance. The smaller the a^k , the more credible the current knowledge is. The larger the arousal rate, the more doubtful is the current

Elton Lima and Fernando Gomide are with Faculty of Electrical and Computer Engineering, University of Campinas, Campinas, Brazil (gomide@dca.fee.unicamp.br).

Rosangela Ballini is with the Institute of Economics, University of Campinas, Campinas, Brazil (email: ballini@eco.unicamp.br).

knowledge. The expression to compute a^k suggested in [1] is the following

$$a^{k+1} = a^k + \beta(1 - \rho^{k+1} - a^k) \quad (4)$$

where $\beta \in [0, 1]$ is a learning rate, $k = 1, 2, \dots$

Figure 1 overviews the participatory learning process and the interplay between the compatibility and arousal mechanisms.

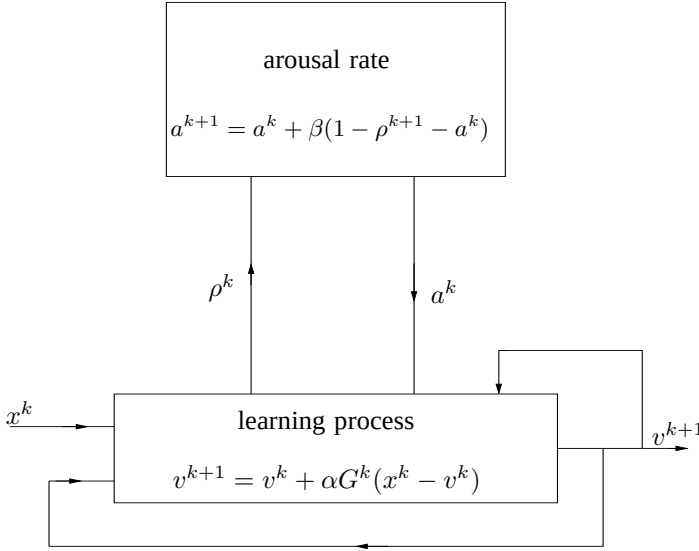


Fig. 1. Participatory learning process.

The complete participatory learning [6] chooses $G^k = (\rho^k)^{1-a^k}$ which gives the prototypical participatory learning process

$$v^{k+1} = v^k + \alpha(\rho^k)^{1-a^k}(x^k - v^k) \quad (5)$$

III. DYNAMIC ANALYSIS OF PARTICIPATORY LEARNING

In what follows we assume, without loss of generality, that $p = 1$. Let us also suppose that the inputs $x^k = f(v^k)$, that is, values x^k , $k = 1, 2, \dots$ are a continuous function $f : [0, 1] \rightarrow [0, 1]$ of the learning variable. In this case, the dynamics of the participatory learning process can be expressed by

$$v^{k+1} = v^k + \alpha^k G^k(f(v^k) - v^k) \quad (6)$$

Since function f is continuous in $[0, 1]$, there exists at least a point $\bar{v} \in [0, 1]$ such that $f(\bar{v}) = \bar{v}$, that is, $\bar{v}$ is a fixed point of f [7]. Indeed, if we let $h(v) = f(v) - v$, then $h : [0, 1] \rightarrow [-1, 1]$ is also continuous and, because the origin $0 \in [-1, 1]$, from the intermediate value theorem [7] we know there exists $\bar{v} \in [0, 1]$ such that $h(\bar{v}) = 0$.

Assume the participatory learning process expressed by (6).

Lemma 1: Let $f : [0, 1] \rightarrow [0, 1]$ be a continuous function and $I \subseteq [0, 1]$ an interval in which $f'(v) = \frac{df(v)}{dv}$ also is continuous for $\bar{v} \in I$. If $f'(v) < 0$ in I , then

$\lim_{k \rightarrow \infty} v^k = \bar{v}$ if $\alpha^k \in (0, 1]$, that is, the sequence $\{v^k\}$ produced by (6) converges to a fixed point of f .

Proof: Let $v^1 \in I$; two cases are of interest (except the obvious case $v^1 = \bar{v}$):

- (i) $f(v^1) - v^1 > 0$
- (ii) $f(v^1) - v^1 < 0$

(i): In the first case $f(v^1) - v^1 > 0$ and for $\alpha^k G^k \in [0, 1]$ we have:

$$\begin{aligned} k = 1 : & \quad v^2 = v^1 + \alpha^1 G^1(f(v^1) - v^1) > v^1; \\ k = 2 : & \quad v^3 = v^2 + \alpha^2 G^2(f(v^2) - v^2) > v^2; \\ & \quad \vdots \quad v^{k+1} > v^k \end{aligned}$$

which suggests, as we show below (see also Figure 2), that $(f(v^k) - v^k)$ does not change sign. Thus, $\{v^k\}$ is a limited, monotone increasing sequence and hence there is a $\bar{v}$ such that $\lim_{k \rightarrow \infty} v^k = \bar{v}$ [7].

Knowing that the sequence $\{v^k\}$ has a limit, to show that $f(\bar{v}) = \bar{v}$, namely, that $\bar{v}$ is a fixed point, it suffices to note from (6) that

$$\lim_{k \rightarrow \infty} v^{k+1} = \lim_{k \rightarrow \infty} v^k + \lim_{k \rightarrow \infty} \alpha^k G^k(f(v^k) - v^k)$$

$$\lim_{k \rightarrow \infty} \alpha^k G^k(f(v^k) - v^k) = 0$$

and, because $\alpha^k G^k$ is by definition finite, $\lim_{k \rightarrow \infty} (f(v^k) - v^k) = 0$.

Therefore, once f is continuous we have

$$f(\lim_{k \rightarrow \infty} v^k) = \lim_{k \rightarrow \infty} v^k \Rightarrow f(\bar{v}) = \bar{v}$$

which means that $\bar{v} = \sup_k \{v^k\}$. In other words, the fixed point is the upper bound of the sequence $\{v^k\}$ [7].

Next, we need to show that $(f(v^k) - v^k)$ indeed does not change sign in I . Suppose there is a j for which $(f(v^j) - v^j) > 0$ and $(f(v^{j+1}) - v^{j+1}) < 0$. In this case

$$f(v^j) - f(v^{j+1}) - v^j + v^{j+1} > 0$$

$$\frac{f(v^j) - f(v^{j+1})}{v^j - v^{j+1}} > 1 > 0$$

From the mean value theorem [7], there is a $v^* \in (v^j, v^{j+1})$ with $f'(v^*) > 1 > 0$, which contradicts the assumption that $f'(\bar{v}) < 0$ because the fixed point $\bar{v} \in (v^j, v^{j+1})$. Therefore, we conclude that $(f(v^k) - v^k)$ does not change sign for $k = 1, 2, \dots$

(ii): The proof when $f(v^1) - v^1 < 0$ is similar. Note, however, that in this case $\{v^k\}$ is a limited, monotone decreasing sequence with fixed point $\bar{v} = \inf_k \{v^k\}$, that is, the fixed point is the lower bound of $\{v^k\}$. ■

When $\bar{v}$ is a fixed point, but $f'(\bar{v}) > 0$, the conditions for convergence of the sequence $\{v^k\}$ are as given by the following lemma.

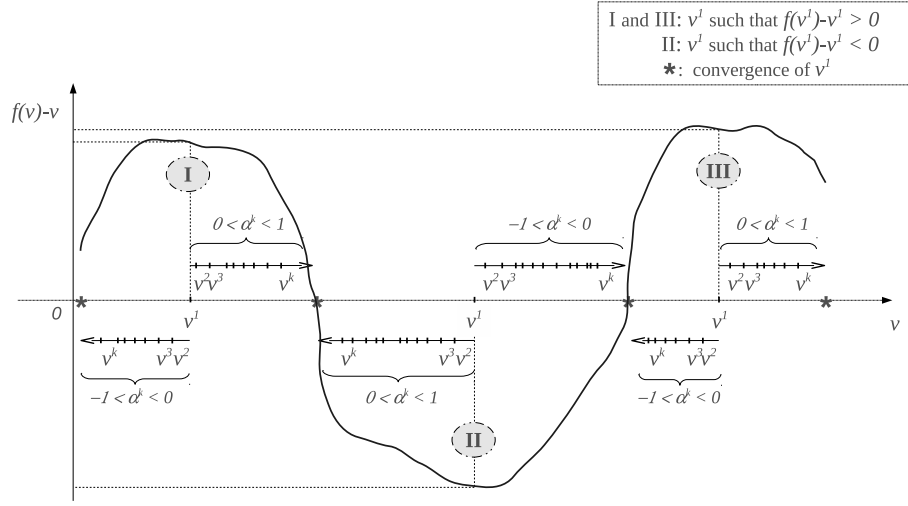


Fig. 2. Behavior of the sequence $\{f(v^k) - v^k\}$ for different initial values v^1 .

Lemma 2: Consider the hypothesis of Lemma 1 and $f'(\bar{v}) > 0$ in I . Then, the sequence $\{v^k\}$ produced by the participatory learning process (6) is such that $\lim_{k \rightarrow \infty} v^k = \bar{v}$ when $\alpha^k \in [-1, 0)$.

Proof: The proof is similar to the proof of Lemma 1. Just note that, in this case, $\{v^k\}$ is monotone increasing if $f(v^1) - v^1 < 0$ and decreasing if $f(v^1) - v^1 > 0$. ■

The next result is a characterization of the convergence of the sequence $\{v^k\}$ induced by (6), whenever a fixed point exists.

Theorem 1: Let $f : [0, 1] \rightarrow [0, 1]$ and $I \subseteq [0, 1]$ an interval in which $\frac{df(v)}{dv}$ exists and the fixed point $\bar{v} \in I$. Then the sequence $\{v^k\}$ produced by (6) converges if $\alpha^k \neq 0$ and $|\alpha^k| \leq 1$.

Proof: Immediate from Lemmas 1 and 2 because either $\alpha^k \in [-1, 0)$ or $\alpha^k \in (0, 1]$ which means $|\alpha^k| \leq 1$. ■

The signal of the sequence $\{f(v^k) - v^k\}$ is the same as of the initial value $f(v^1) - v^1$, as Lemma 1 shows. See Figure 2 for an illustration. The fixed point depends on the sign of α^k . Lemmas 1 and 2 show that $\{v^k\}$ is a monotone and convergent sequence. A simple example is given in Figure 3 for $f(v) = \sin(v) + v$.

As suggested by Figure 2 and summarized by Theorem 1, if $0 < \alpha^k < 1$, then the fixed point when the initial value v^1 is in region I is the same when v^1 is in II. Similarly, when $-1 < \alpha^k < 0$, the fixed point $\{v^k\}$ is the same if v^1 is in region II or III. Figure 2 also stress that $f(v^1) - v^1$ and $\{f(v^k) - v^k\}$ always have the same sign.

Summing up, the results above give conditions for which the the sequence of values $\{v^k\}$ produced by participatory learning expressed by (6) converges when inputs are a continuous and differentiable function of the learning variable. Continuity and differentiability of f are reasonable requirements in many practical applications. It remains to explain

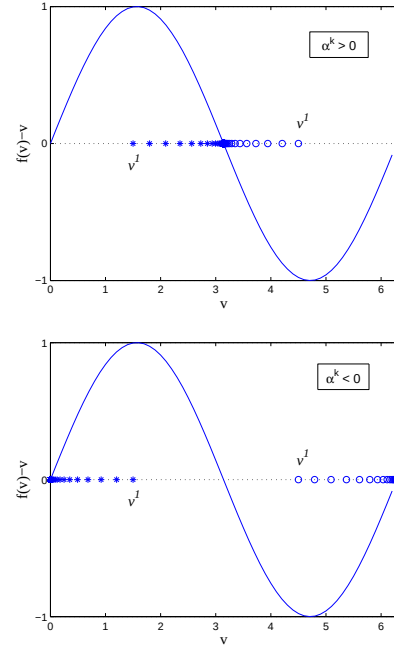


Fig. 3. Convergence of the participatory learning process.

how the learning algorithm converges. Next we characterize the convergence rate of $\{v^k\}$, namely, the speed with which $v^k \rightarrow \bar{v}$.

Let us recall that a sequence $\{v^k\}$ converges to a value $\bar{v}$ in order $q > 0$ if there exists a positive, real-valued constant K such that [8]

$$|v^{k+1} - \bar{v}| \leq K |v^k - \bar{v}|^q$$

Theorem 2: Consider the same hypothesis of Theorem 1. If $\lim_{k \rightarrow \infty} v^k = \bar{v}$ exists, then there exists a neighborhood of $\bar{v}$ in which $\{v^k\}$ converges in order $q = 1$ (linear convergence).

Proof: Consider $h(v) = (f(v) - v)$, $\theta^k = (v^k - \bar{v})$ and

$\alpha^k G^k \in [0, 1]$. From expression (6) for $\{v^k\}$ we have

$$v^{k+1} = v^k + \alpha^k G^k h(v^k)$$

$$\theta^{k+1} = \theta^k + \alpha^k G^k h(v^k) \leq \theta^k + h(v^k)$$

We must show that there exists a positive real-valued constant K such that $|\theta^{k+1}| \leq K|\theta^k|$. For all v^k close to $\bar{v}$, there exists a ξ between v^k and $\bar{v}$ such that (mean value theorem)

$$h'(\xi) = \frac{h(v^k) - h(\bar{v})}{v^k - \bar{v}} = \frac{h(v^k) - 0}{\theta^k}$$

$$h(v^k) = h'(\xi)\theta^k$$

Therefore $\theta^k + h(v^k) = [1 + h'(\xi)]\theta^k$ and, assuming $h'(\xi)$ bounded, there exists a positive constant K with $|h'(\xi)| < K$. Hence

$$|\theta^{k+1}| \leq |[1 + h'(\xi)]\theta^k| \leq (1 + K)|\theta^k|.$$

■

We see that $\{v^k\}$ converges in order $q = 1$, or equivalently, the sequence $\{v^k\}$ converges linearly to fixed point $\bar{v}$ whenever it exists. Generally speaking, learning processes such as (6) converge linearly because they use zero-order information of f only, that is, they need just the value of f at each step.

Notice that Theorems 1 and 2 provide necessary conditions only. There may exist situations in which f is neither differentiable nor continuous, but the sequence $\{v^k\}$ may still converge.

IV. CONCLUSIONS

In this paper we have studied the dynamic behavior of the participatory learning model. The study assumes that learning data is a function of the object of learning. A characteristic of this learning model is the variable compatibility and arousal index that involves variable learning rates. This feature raises the important problem of proving the convergence of the learning system for this condition. Necessary conditions have been developed considering the joint effect of compatibility, arousal and learning rates. Future work shall address the general case, the situation when no assumption is made on how the learning variable is related with the input data.

V. ACKNOWLEDGMENTS

The first author acknowledges FAPESP, the Research Foundation of the State of São Paulo for fellowship 05/58879-1. The last authors thank the Brazilian National Research Council, CNPq, for grants 302407/2008-1 and 304857/2006-8, respectively.

REFERENCES

[1] R. Yager, "A model of participatory learning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 20, no. 05, pp. 1229–1234, September/October 1990.

[2] L. Silva, F. Gomide, and R. Yager, "Fuzzy clustering with participatory learning and applications," in *Advances in Fuzzy Clustering and its Applications*, J. Oliveira and W. Pedrycz, Eds., pp. 139–153. Wiley, N.J., USA, 2007.

[3] P. Angelov, D. P. Filev, and N. Kasabov, *Evolving Intelligent Systems: Methodology and Applications*, Wiley-IEEE Press, March 2010.

[4] E. Lima, M. Hell, R. Ballini, and F. Gomide, "Evolving fuzzy modeling using participatory learning," in *Evolving Intelligent Systems*, P. Angelov, D. Filev, and N. Kasabov, Eds., chapter 4, pp. 67–86. Wiley, N.J., USA, 2010.

[5] R. Yager and D. Filev, "Modeling participatory learning as a control mechanism," *International Journal of Intelligent Systems*, , no. 08, pp. 431–450, 1993.

[6] R. Yager, "Participatory learning: A paradigm for more human like learning," in *2004 IEEE International Conference on Fuzzy Systems, FUZZ-IEEE*, Budapest, Hungary, July 2004, pp. 79–84.

[7] R. Bartle, *The Elements of Real Analysis*, Wiley, NY, EUA, 2nd edition, 1976.

[8] S. Conte and C. de Boor, *Elementary Numerical Analysis - An Algorithmic Approach*, McGraw-Hill, NY, EUA, 3rd edition, 1981.

Incremental Classification of Images by Human Assisted Fuzzy Similarity Analysis

Gancho Vachkov, *Member, IEEE*

Abstract—This paper proposes a computational scheme for incremental classification of images, based on human assisted fuzzy similarity analysis. First of all, two main parameters from each image are extracted in the form of a *center-of-gravity* and a *generalized volume* of the image model. Then their differences for each pair of images are taken as respective features $F1$ and $F2$, which serve as inputs of the fuzzy inference system for similarity analysis. This system uses special *asymmetrical* Gaussian membership functions that are later tuned by using a predefined list of *human decisions* (similarities). The list includes fixed number of available pairs of images and the objective is to minimize the discrepancy between the human and the computer similarity decision.

The proposed incremental classification scheme uses an initial list of *Core Images* to compare the new sequentially coming images with the core images. With a predetermined threshold, some images are judged as members of the *existing class* in the Core and other form new members that create a *new class* in the Image Base.

The flexibility and applicability of the proposed human assisted incremental classification is illustrated on an example for classification of 16 flower images and the results are discussed in the paper.

I. INTRODUCTION

Clustering and classification problems [1,2] that usually are dealing with a large number of objects use different kinds of similarity analysis of these objects in order to obtain the solution. A typical example is the similarity analysis, performed over a large amount of different images, which is important step in the computational procedure for classification of pictorial information. Similarity analysis and classification of pictorial information is a very specific area of activity, where in most case the human expert is superior to the available computerized systems. Among the different reasons for such situation are the complexity and the subjective nature (vagueness) of the problem itself, as well as the variety of the possible criteria that could be used for the similarity analysis. Despite such difficulties, finding a “good”, reasonable solution to the problem of similarity evaluation is a key factor for a subsequent plausible classification that leads to many successful applications. Among them are such as automated visual search through a large amount of image information, its proper sorting and classification for the purpose of different kinds of decision making and (technical or medical) diagnosis.

Gancho Vachkov is with the Department of Reliability-based Information Systems Engineering, Kagawa University, Hayashi-cho 2217-20, Takamatsu-shi, Kagawa-ken 761-0396, Japan (Phone: +81-87-864-2265; Fax: +81-87-864-2262; Email: vachkov@eng.kagawa-u.ac.jp).

In this paper we present a computational scheme for incremental type of human assisted classification of images. Here a fuzzy inference procedure for similarity analysis is used, which can learn the human expert experience in order to properly adjust the decision process. The incremental nature of the proposed computation scheme leads to creating a growing type of *image base*, which in fact accumulates past knowledge about the similarity and decision making.

All the details as well as an example for classification of 16 flower images are given in the sequel of the paper.

II. INCREMENTAL CLASSIFICATION SCHEME BASED ON FUZZY SIMILARITY ANALYSIS

The block diagram of the proposed incremental classification scheme is presented in Fig. 3.1., which is a further development of our previous ideas for similarity analysis, discussed in [7,8].

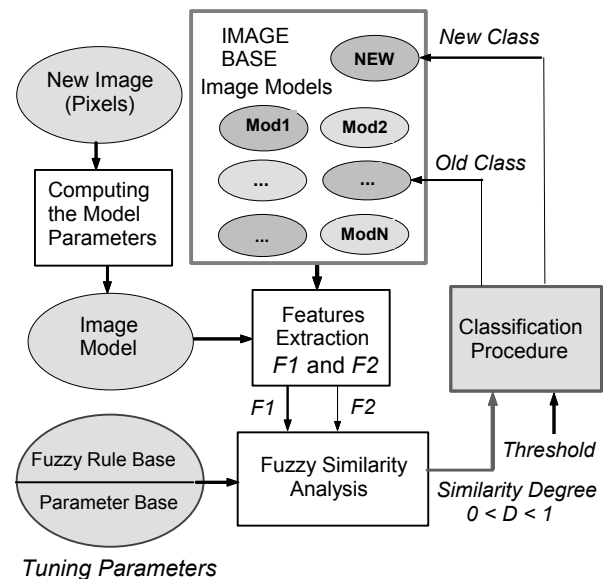


Fig. 2.1. Block Diagram of the proposed Incremental Classification scheme, based on Fuzzy Similarity Analysis.

The procedure starts with a small number of known *core images* which form the initial size of the *Image Base*. During classification, the similarity degree of each new (unknown) image is computed against all currently existing *core images* in the *Image Base*. As a result, depending on the preliminary given *threshold* Th for classification, the new image could join an *existing class* (core image) in the current *Image Base* or could be classified as “different image” thus forming a *new class* and thus *enlarging* the *Image Base*. In such way, the proposed classification is an incremental procedure, allowing

the Image Base to gradually grow, when a new image with a low similarity to all currently existing core images is discovered.

From Fig. 2.1. it is seen that the raw data (pixels) from each image are used to derive a simple *Image Model* that contains two parameters (discussed in the next Section). Then for each pair of images to be compared, two respective features *F1* and *F2* are calculated and used as inputs of the block for fuzzy similarity analysis. Finally the similarity degree *D* is used for the incremental classification procedure. All computational details are given in Sections III and IV of the paper.

III. IMAGE MODELS PARAMETERS AND FEATURES EXTRACTION FOR SIMILARITY ANALYSIS

Quite often the first step before the actual similarity analysis and classification of the images is to reduce the large amount of the “raw pixel” (RGB) information contained in the original images. This step is called *Information Compression* and has been discussed previously in the literature [4-8].

From a computational viewpoint the information compression could be considered as a *transformation* of the original large data set: $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{iK}]$, $i = 1, 2, \dots, M$, consisting of *M* pixel data in the $K=3$ dimensional input space (RGB). This *information compression* procedure can be performed by using different unsupervised competitive learning algorithms, such as clustering algorithms [1,2], the Self-Organizing (Kohonen) Maps [3], the Neural-Gas Algorithm [4] and other competitive learning algorithms [5,6]. The common point here is that all these algorithms try to find the most appropriate positions the *N* neurons (clusters) in the *K*-dimensional parameter space so that to resemble as much as possible the density distribution of the original data in the space.

Such compression is worth for a very large number of data and have been used in our previous works as in [7,8]. However, for *small* to *medium* number of data (in the approximate range of about 200-300,000 pixels or less), the *direct computation* of all “raw data” could be a preferable option. Further on in this paper we use such the *direct computation* method for calculating the Image Model parameters.

We propose here the use of the following two Image Model parameters P1 and P2 that can be calculated directly from the “raw pixel data” as follows:

1) *The Center-of-Gravity*: $P1 = \mathbf{CG} = [CG_1, CG_2, \dots, CG_K]$ of the $K = 3$ dimensional data is a vector, with its three components for R, G and B computed as follows:

$$CG_j = \frac{\sum_{i=1}^M x_{ij}}{M}, j = 1, 2, \dots, K \quad (1)$$

2) *The Generalized Volume* $P2 = VOL$ of the Image Model:

$$P2 = VOL = 2 \sqrt{\sum_{j=1}^K R_j^2} \quad (2)$$

where

$$R_j = \frac{1}{M} \sum_{i=1}^M \sqrt{(x_{ij} - CG_j)^2}, j = 1, 2, \dots, K \quad (3)$$

It is seen from above equations that the volume P2 is a kind of measure of the *size* of the 3-dimensional oval (ellipsoid) of the *pixel cloud* for this image.

According to the block diagram in Fig. 2.1. the next step is to extract the features *F1* and *F2* from a given pair of images (*A, B*) that are further used as *Inputs* of the *fuzzy similarity* analysis. They are computed in a very simple way as follows:

- 1) The Feature *F1* is the Center-of-Gravity Distance *CGD* between Image *A* and Image *B*:

$$F1 \equiv CGD_{AB} = \sqrt{\sum_{j=1}^K [CG_j^A - CG_j^B]^2} \quad (4)$$

- 2) The Feature *F2* is called Volume Difference *VLD* between Image *A* and Image *B*:

$$F2 \equiv VLD_{AB} = |VOL_A - VOL_B| \quad (5)$$

IV. BASIC STRUCTURE OF THE FUZZY RULE BASED SIMILARITY ANALYSIS

The next computation step, as seen from the block diagram in Fig. 2.1., is the *Fuzzy Similarity* Analysis. It is basically a standard *fuzzy inference* procedure, which consists of the following three main computation steps:

- 1) *Fuzzification*;
- 2) *Fuzzy Inference* (with *Product Operation*) and
- 3) *Defuzzification* (*Weighted Average*).

As for the membership functions, we use here a smooth type *Asymmetrical Gaussian* functions with *different* width parameters σ_L and σ_R for the left and the right part respectively. There are two main reasons for such choice. The first one is that the smooth Gaussian functions can better represent the nonlinearity in the fuzzy decision. The second one is that by selecting different width parameters we are able to keep a “good” enough and *constant* overlapping level between any pair of *neighboring* membership functions. Thus more plausible results could be obtained and additionally, a reduction of the number of tuning parameters in the next optimization task.

An illustrative example of the proposed asymmetrical type Gaussian membership functions is given in Fig. 4.1. with three different predetermined overlapping levels. It has been found empirically that the left and the right width for a given membership function with a center at *C* can be calculated as follows:

$$\sigma_L = R(C - CL); \quad \sigma_R = R(CR - C) \quad (6)$$

Here *R* is the user-defined *overlapping ratio* and *CL* and

CR are the *centers* of the *left* and *right* membership function, respectively.

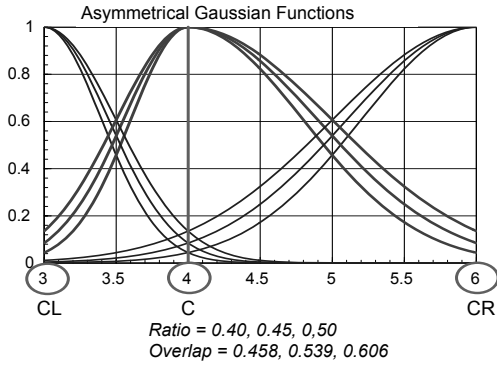


Fig. 4.1. The assumed Asymmetrical Gaussian Membership Functions .

Further on we use 5 membership functions for the features $F1$ and $F2$, with the following linguistic labels:

$VS = \text{Very Small}$; $SM = \text{Small}$; $MD = \text{Medium}$; $BG = \text{Big}$ and $VB = \text{Very Big}$;

Then the Fuzzy Rule Base (FRB) for the Fuzzy Inference procedure consists of 25 fuzzy rules, as shown in Fig. 4.2.

		D Fuzzy Difference between two Images				
F2 ↑ Volume Difference	VB	U5	U6	U7	U8	U9
	BG	U4	U5	U6	U7	U8
	MD	U3	U4	U5	U6	U7
	SM	U2	U3	U4	U5	U6
	VS	U1	U2	U3	U4	U5
		VS	SM	MD	BG	VB
		Center-of-Gravity Distance → F1				

Fig. 4.2. The Fuzzy Rule Base used for the Similarity Analysis.

Each fuzzy rule has its crisp numerical output (*Singleton*) that is selected among the available full set of 9 singletons $U_1, U_2, \dots, U_9$, with $U_1 < U_2 < \dots < U_9$. These singletons represent the *Difference Degree* (or *Dissimilarity Degree*) $0.0 \leq D \leq 1.0$ for the features $F1$ and $F2$ computed for a given pair of Images (A, B).

The proposed *Fuzzy Rule Base* $D = F(F1, F2)$ in Fig. 4.2. is constructed by using basic *human logic* and understanding about similarity between pairs of images. .

The numerical values of all 9 *singletons* have been fixed in this paper, as follows:

$$\begin{aligned} U_1 &= 0.0; & U_2 &= 0.125; & U_3 &= 0.250; \\ U_4 &= 0.375; & U_5 &= 0.500; & U_6 &= 0.625; \\ U_7 &= 0.750; & U_8 &= 0.875; & U_9 &= 1.0 \end{aligned} \quad (7)$$

As an example, the crisp output of the Fuzzy Rule No. 14 ($i = 14$) from Fig. 4.2. will be :

$$\begin{aligned} \text{IF}(P1 \text{ is MD AND } P2 \text{ is BG}) \text{ THEN} \\ u_{14} = U_6 = 0.625 \end{aligned} \quad (8)$$

It is clear from Fig. 4.2. that all fuzzy rules have their individual *Singletons*: $u_i \in [U_1, U_2, \dots, U_9]$, $i = 1, 2, \dots, L$.

The standard weighted-average method for *Defuzzification* is used for computing the *Dissimilarity Degrees* D between a given pair of images, as follows:

$$D = \frac{\sum_{i=1}^L u_i v_i}{\sum_{i=1}^L v_i} \quad (9)$$

Here $0 \leq v_i \leq 1$, $i = 1, 2, \dots, L$ is the *Firing (Activation) Degree* of the i -th fuzzy rule and $L = 25$ is the total number of the fuzzy rules.

The above proposed incremental classification scheme, based on fuzzy similarity is illustrated here on an example of 16 images of different flowers, as shown in Fig. 5.1.





Fig. 5.1. An example of 16 Images of flowers, used for similarity analysis and Incremental Classification.

As seen from this figure, some images look very *similar*, while others are quite *different*, if taking into account the variety of colors and the overall (vague) human evaluation.

First, the BMP files with the original images have been preprocessed in order to obtain the respective 3-dimensional “raw data”, which contain the *pixel values* (the 3-dimensional matrices: *R-G-B*) for each image, within the range of 0-255.

Each image has a resolution of 256×196 , which makes $M = 49,152$ pixels for further direct processing. The pixels form a kind of *cloud* in the RGB space, as depicted for a visual comparison of two images in Fig. 5.2.

The center-of-gravities *COG* of all 16 Images from Fig. 5.1 are computed by (1) and shown in the 3-dimensional RGB space in Fig. 5.3. As seen from this figure, some of them are quite close (obviously corresponding to similar images) while others are far from each other (possibly for different images).

The next Fig. 5.4. shows the image model volumes *VOL* for all 16 images from Fig. 5.1. As seen there is a big difference between some images, which provides valuable information for the next classification procedure.

The above two figures Fig. 5.3. and Fig. 5.4. show the complexity of the problem for similarity analysis, which is obviously difficult to be solved if using only each of the features *F1* or *F2* separately.

Fig. 5.5. is a two-dimensional plot of the features *F1* and *F2* for all 120 pairs of images. It is helpful for approximate estimation of the *ranges* $[min, max]$ of the features *F1* and *F2*. These ranges are utilized to fix the *borders* for the *initial positions* of the asymmetrical membership functions, as shown in Fig. 5.5. The computed 3-D response surface of the initial fuzzy inference system is shown in Fig. 5.6.

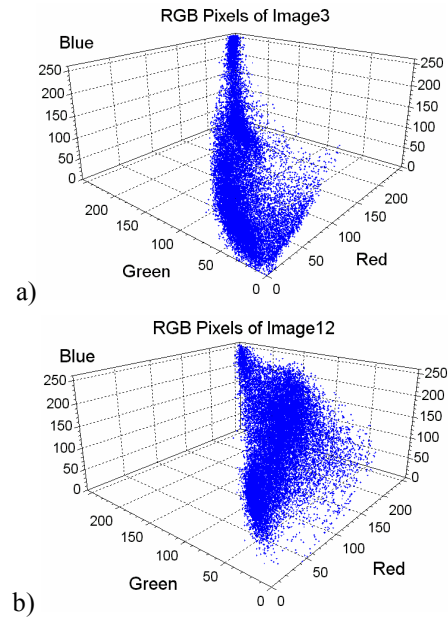


Fig. 5.2. 3-Dimensional plot of the Raw Data (RGB Pixels) of *Image3* and *Image12* from Fig. 5.1. They form different “clouds of data” in the space.

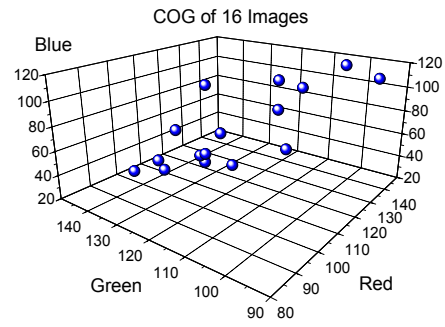


Fig. 5.3. Center-of-Gravities COG, computed by (1) for all 16 Images.

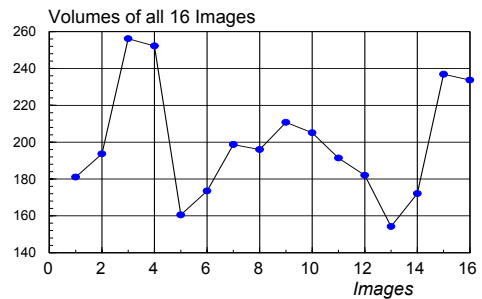


Fig. 5.4. The Generalized Volumes *VOL* of all 16 Images from Fig. 5.1.

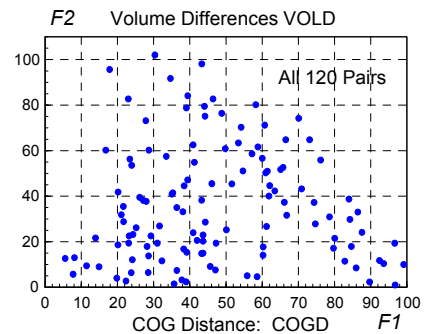


Fig. 5.5. Locations of the Features (*F1*, *F2*) for all 120 pairs of Images.

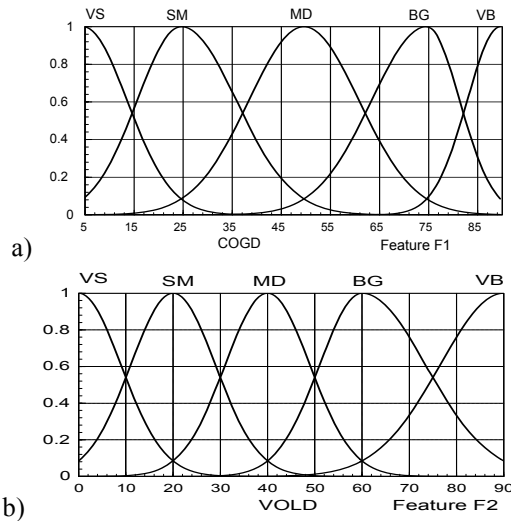


Fig. 5.6. Initial positions of the membership functions for $F1$ and $F2$.

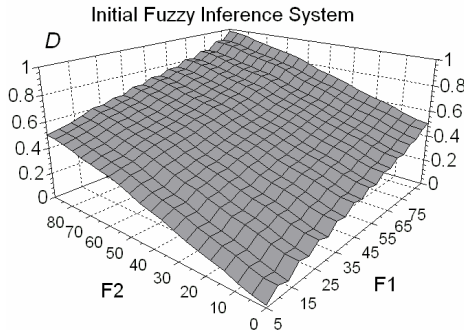


Fig. 5.7. The response surface of the initial fuzzy decision system.

V. THE HUMAN ASSISTED FUZZY SIMILARITY ANALYSIS

It is obvious that plausible “human-like” results from the Fuzzy similarity analysis could be obtained only by proper tuning of the membership functions in Fig. 5.6., as well as the singleton parameters in (7). Therefore, in order to make the computer-based similarity analysis closer to the human decision, we have chosen a set of 14 pairs of images, for which an appropriate *human decision* is available, s shown in Fig. 6.1. All the human decisions are arranged in *four groups* (4 rough levels): **0.1**; **0.3**; **0.5** and **0.7**, as will be shown later in Fig. 6.4.

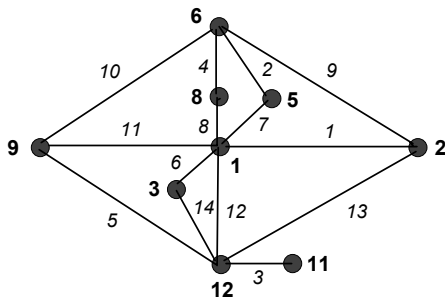


Fig. 6.1. Graph structure of all images. The selected pairs for human decision are shown as arcs between the respective nodes (images).

It is clear that the human decisions are *subjective* ones, as is also the choice of the *number* of selected pairs. Therefore all

these parameters affect the results from the computer system for similarity analysis.

By using the above assumed 14 human decisions, an appropriate tuning of the Fuzzy Inference System has to be performed in order to minimize the *Discrepancy* (the error) between the *human based* and the *computer based* decision for similarity. This discrepancy is calculated here in a simple way, as the *mean* of the absolute differences between all 14 decisions.

In principal, any multi-variable optimization procedure could be applied for tuning the fuzzy inference system. In this paper we have used a modified version of the popular Particle Swarm Optimization (PSO) algorithm with *Inertia Weight* [9]. This algorithm optimizes the *intermediate* locations of the membership functions, i.e. the variables SM , MD and BG for the features $F1$ and $F2$. This is in fact a 6-dimensional optimization with special implemented constrains: $SM < MD < BG$. Because of space limitations, further details are omitted in this paper.

The optimized membership functions are shown in Fig. 6.2. and the resulted 3-D *nonlinear* response function is shown in Fig. 6.3.

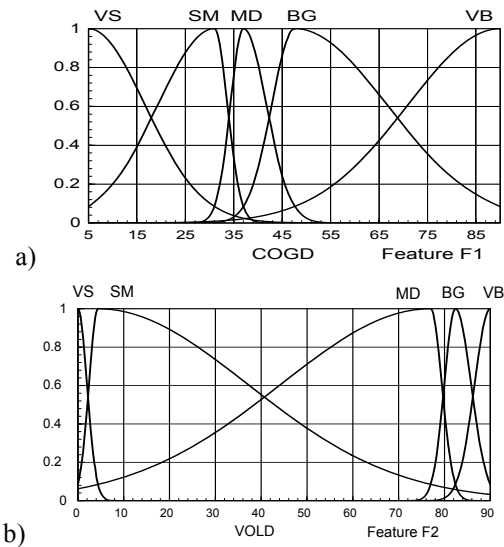


Fig. 6.2. The optimized Membership Functions for $F1$ and $F2$.

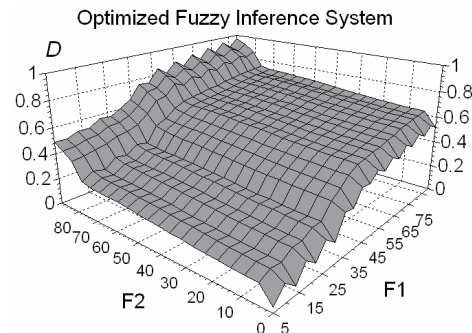


Fig. 6.3. The optimized Nonlinear Response Surface of the Fuzzy Decision.

The results of the similarity analysis and the Discrepancy before and after the optimization are shown in Fig. 6.4.

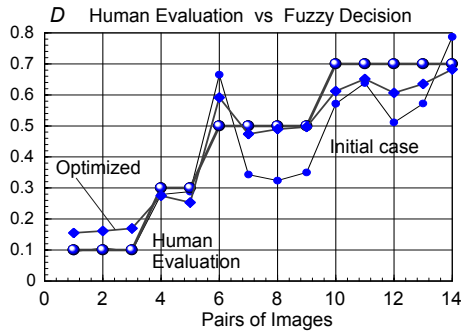


Fig. 6.4. Results from optimization of the computer-based similarity analysis by using a predetermined set of 14 human decisions from Fig. 6.1.

The mean *Discrepancy* level has decreased from 0.0921 in the initial (non-tuned) case to 0.0499 after optimization. The improvement can be visually noticed in Fig. 6.4.

VI. RESULTS FROM THE INCREMENTAL CLASSIFICATION

As seen from Fig. 2.1 in Section II, before performing the incremental classification, we assume that an *initial* Image Base with a small number of *Core Images* exists, in order to make the further comparisons with the new coming images.

It is natural to expect that the initial choice of the number and the list of the core images would affect the results from the online classification of the newly submitted images. The predefined *threshold* Th from Fig. 2.1. also affects the results from the classification.

At each step of the classification procedure, one new image q (different from the list of core images) is submitted for evaluation. Then the *so called minimal dissimilarity* $D_{\min}^p$ is calculated by (9) between the new image q and a certain core image p from the Image Base. If $D_{\min}^p \leq Th$, a decision is made that the new image q belongs to the class p in the Image Base. Otherwise, image q creates a *new class* thus enlarging the Image Base by one new member.

Results from the proposed online incremental classification procedure is shown in Table 1 by using four different sets of *core images*, as seen in the table.

Table 1. 4 Examples of Online Incremental Image Classification

Step	Core: 1, 9	Core: 5, 9	Core: 1, 3	Core 1, 3, 9
1	2 1	1 1	2 1	2 1
2	3 3	2 1	4 3	4 3
3	4 3	3 3	5 5	5 5
4	5 5	4 3	6 5	6 5
5	6 5	6 5	7 7	7 9
6	7 9	7 9	8 7	8 5
7	8 5	8 5	9 7	10 9
8	10 9	10 9	10 7	11 9
9	11 9	11 9	11 11	12 9
10	12 9	12 9	12 11	13 5
11	13 5	13 5	13 5	14 1
12	14 1	14 1	14 1	15 9
13	15 9	15 9	15 7	16 3
14	16 3	16 3	16 3	-

A threshold $Th = 0.3$ is assumed for all classifications. For each assumed set of *core images*, all the remaining images (from the whole group of 16 images in Fig. 5.1.) are considered as *new images* and submitted sequentially *one-by-one*, as seen from Table 1. The **bold numbers** in each column of the table show the class, to which the new image (the left number) belongs. During classification, the following *new classes* have been added in all 4 examples:

- for *Core* {1, 9} – new classes are: **3** and **5**;
- for *Core* {5, 9} – new classes are: **1** and **3**;
- for *Core* {1, 3} – new classes are: **6**, **7** and **11**;
- for *Core* {1, 3, 9} – new classes is **5**;

VII. DISCUSSIONS AND CONCLUSIONS

An incremental semi-unsupervised classification scheme for images is proposed in this paper. It uses a human expert experience for initial tuning of the fuzzy similarity system, before the actual incremental classification. The scheme is applicable for gradual incremental knowledge acquisition from large (unconstrained) number of images.

An important feature of the system is that its performance resembles closely the decision making of a particular human expert, used for its tuning. Therefore the tuned system could be regarded as a *decision making agent* for similarity analysis and classification. Then, a multi-agent system for a “*team decision making*” could be a possible option for further development of the proposed idea.

There are some problems to be solved in future for improvement of this classification system. One is, for example, to find a good way for selecting of the most appropriate *initial core images* in the Image Base. Other problem is to find an appropriate algorithm for *pruning* and *periodically refreshing* (re-tuning) of the Image Base, based on the additional available information.

REFERENCES

- [1] Bezdek, J.C., *Pattern Recognition with Fuzzy Objective Function Algorithms*, New York: Plenum Press, 1981.
- [2] Bishop Ch.M., *Neural Networks for Pattern Recognition*. Oxford University Press, 2003.
- [3] Kohonen, T., *Self-Organizing Maps*, Third Edition, Springer Series in Information Sciences, Springer, Berlin, 2001.
- [4] Martinetz T., Berkovich S., and Schulten, K., “Neural-Gas Network for Vector Quantization and Its Application to Time-Series Prediction”, *IEEE Trans. Neural Networks*, Vol. 4, No. 4, pp. 558-569, 1993.
- [5] Zhang, Ya-Jun and Liu, Zhi-Qiang, “Self-Splitting Competitive Learning: A New On-Line Clustering Paradigm”, *IEEE Trans. on Neural Networks*, Vol. 13, No. 2, pp. 369-380, 2002.
- [6] Xu, L., Krzyzak A. and Oja, A., “Rival Penalized Competitive Learning for Clustering Analysis, RBF Net and Curve Detection”, *IEEE Trans. Neural Networks*, Vol. 4, No. 4, pp. 636-649, 1993.
- [7] Vachkov, G., “Human Assisted Fuzzy Image Similarity Analysis Based on Information Compression”, *Journal of Advanced Comp. Intelligence and Intelligent Informatics*, JACIII, vol. 13, no. 3, pp. 255-261, 2009.
- [8] Vachkov, G., “On-Line Unsupervised Learning for Information Compression and Similarity Analysis of Large Data Sets”, CD-ROM Proceedings of the 2007 IEEE Int. Conference on Mechatronics and Automation, ICMA 2007, Harbin, China, pp.105-110, Aug. 5-8, 2007.
- [9] Poli, R., Kennedy, J. and Blackwell, T., “Particle Swarm Optimization. An Overview”, *Swarm Intelligence*, vol. 1, pp. 33-57, 2007.

Clustering to train an evolving radial basis function

European Conference on Artificial Intelligence

Jose de Jesus Rubio¹ and Jaime Pacheco¹ and Raul Rivera¹

Abstract. In this paper, we propose the backpropagation algorithm to train online an evolving radial basis function. Structure and parameter learning are updated at the same time in our algorithm, we do not make difference in structure learning and parameter learning. It generate groups with an online clustering. The center is updated in order to get that the center is near to the incoming data in each iteration, in this way, It does not need to generate a new rule in each iteration, i.e., it does not generate many rules and It does not need to prune the rules. We give a time varying learning rate for backpropagation training in the parameters.

1 Introduction

In the last few years, the application of fuzzy neural networks to nonlinear system identification is very active area [10],[11]. Fuzzy modeling involves structure and parameters identification. The second one is usually addressed by some gradient descent variant, e.g., the least square algorithm and backpropagation. The online identification is addressed in this paper.

In on line identification, model structure and parameter identification are updated immediately after each input-output pair has been presented, that is, after each iteration. On line identification also includes; i) model structure and ii) parameter identification. There are a few online methods in the literature. In [7] the input space is partitioned according to a aligned clustering-based algorithm. After the number of rules is decided, the parameters are tuned by recursive least square algorithm, it is called SONFIN. In [8] it is the recurrent case of the above case, it is called RSONFIN. In [13] the input space is automatically partitioned into fuzzy subsets by adaptive resonance theory mechanism. Fuzzy rules that tend to give high output error are split in two, by a specific fuzzy rule splitting procedure. In [9] he proposes that the radius to make clustering updates.

In this paper, we propose the backpropagation algorithm to train online an evolving radial basis function. Structure and parameter learning are updated at the same time in our algorithm, we do not make difference in structure learning and parameter learning. It generate groups with an online clustering. The center is updated in order to get that the center is near to the incoming data in each iteration, in this way, It does not need to generate a new rule in each iteration, i.e., it does not generate many rules and It does not need to prune the rules. We give a time varying learning rate for backpropagation training in the parameters.

2 Fuzzy neural networks for nonlinear identification

Consider following unknown discrete-time nonlinear system:

$$y(k-1) = f[X(k-1)] \quad (1)$$

Where $X(k-1) = [x_1(k-1) \dots x_N(k-1)] = [y(k-2), \dots, y(k-n-1), u(k-2), \dots, u(k-m-1)] \in \mathbb{R}^N$ ($N = n+m$) is the input vector, $|u(k-1)|^2 \leq \bar{u}$, $y(k-1)$ is the output of the plant, f is general nonlinear smooth function $f \in C^\infty$. In the case of [6], we consider the radial basis function neural network as:

$$\begin{aligned} \hat{y}(k-1) &= a(k-1)/b(k-1) \\ a(k-1) &= \sum_{j=1}^M v_j(k-1)z_j(k-1), \\ b(k-1) &= \sum_{j=1}^M z_j(k-1) \\ z_j(k-1) &= \prod_{i=1}^N \exp \left[-\left(\frac{x_i(k-1) - c_{ij}(k-1)}{\sigma_{ij}(k-1)} \right)^2 \right] \end{aligned} \quad (2)$$

Where $x_i(k-1)$ are inputs of system (1), ($i = 1 \dots N$), $c_{ij}(k-1)$ and $\sigma_{ij}(k-1)$ are the centers and the widths of the gaussian function, respectively, ($j = 1 \dots M$), $v_j(k-1)$ is the output of the gaussian function.

3 Structure identification

We use the online clustering to train the structure of the algorithm. Choososing an appropriate number of hidden neurons is important in designing the online clustering for neuro fuzzy systems, because too many hidden neurons result in a complex evolving system that may be unnecessary for the problem and it can cause overfitting [6], whereas too few hidden neurons produce a less powerful neural system that may be insufficient to achieve the objective. We view the number of hidden neurons as a design parameter and determine it based on the input-output pairs and on the number of elements of each hidden neuron.

The basic idea is to group the input-output pairs into clusters and use one hidden neuron for one cluster as in [9], [14]; that is, the number of hidden neurons equals the number of clusters.

One of the simplest clustering algorithms is the nearest neighborhood clustering algorithm. In this algorithm, first we put the first data as the center of the first cluster. Then, if the distances of a data to the cluster centers are less than a pre-specified value (the radius r), put this data into the cluster whose center is the closest to this data; otherwise, set this data as a new cluster center. The details are given as follows.

¹ Seccion de Estudios de Posgrado e Investigacion - ESIME Azcapotzalco - Instituto Politecnico Nacional, Mexico, email: jrubioa@ipn.mx

Let $x_i(k-1)$ are newly incoming pattern, then we get:

$$p(k-1) = \max_{1 \leq j \leq M} z_j(k-1) \quad (3)$$

If $p(k-1) < r$, then a new rule is generated (each rule correspond to each center) and $M = M+1$ where r is a selected radius, $r \in (0, 1)$. Once a new rule is generated, the next step is to assign initial centers and widths of the corresponding membership functions.

$$\begin{aligned} c_{i,M+1}(k) &= x_i(k) & v_{M+1}(k) &= y(k) \\ \sigma_{i,M+1}(k) &= rand \in (0, 1) \end{aligned} \quad (4)$$

If $p(k-1) \geq r$, then a rule is not generated and in the case that $z_j(k-1) = p(k-1)$ we have the winner rule j^* , the centers of this rule are updated as:

$$\begin{aligned} c_{ij^*}(k) &= c_{ij^*}(k-1) \\ &+ \frac{1}{1+x_i^2(k-1)+c_{ij^*}^2(k-1)} (x_i(k-1) - c_{ij^*}(k-1)) \end{aligned} \quad (5)$$

4 Parameter identification

We define the identification error as:

$$e(k-1) = \hat{y}(k-1) - y(k-1) \quad (6)$$

If we define:

$$\begin{aligned} D_{1ij}(k-1) &= \frac{2[v_j(k-1) - \hat{y}(k-1)]z_j(k-1)[x_i(k-1) - c_{ij}(k-1)]}{b(k-1)\sigma_{ij}^2(k-1)} \\ D_{2ij}(k-1) &= \frac{2[v_j(k-1) - \hat{y}(k-1)]z_j(k-1)[x_i(k-1) - c_{ij}(k-1)]^2}{b(k-1)\sigma_{ij}^3(k-1)} \\ D_{3j}(k-1) &= \frac{z_j(k-1)}{b(k-1)} \end{aligned} \quad (7)$$

We use the following learning law to updated the weights of neural identifier:

$$\begin{aligned} c_{ij}(k) &= c_{ij}(k-1) \\ &- \eta(k-1)D_{1ij}(k-1)e(k-1) \\ \sigma_{ij}(k) &= \sigma_{ij}(k-1) \\ &- \eta(k-1)D_{2ij}(k-1)e(k-1) \\ v_j(k) &= v_j(k-1) \\ &- \eta(k-1)D_{3j}(k-1)e(k-1) \end{aligned} \quad (8)$$

Where $j = 1 \dots M$, $i = 1 \dots N$ and $D_{1ij}(k-1)$, $D_{2ij}(k-1)$, and $D_{3j}(k-1)$ are given in (7). The dead-zone is applied to $\eta(k-1)$ as:

$$\eta(k-1) = \begin{cases} \frac{\eta}{1+q(k-1)} & \text{if } e^2(k-1) \geq \frac{\zeta^2}{1-\eta} \\ 0 & \text{if } e^2(k-1) < \frac{\zeta^2}{1-\eta} \end{cases} \quad (9)$$

Where $q(k-1) = D_{1ij}^2(k-1) + D_{2ij}^2(k-1) + D_{3j}^2(k-1)$, $0 < \eta \leq 1$.

5 The proposed algorithm

The proposed algorithm is as follows:

0) Select the following parameters: $0 < r < 1 \in \mathbb{R}$, $0 < \eta < 1 \in \mathbb{R}$, $\zeta^2 \in \mathbb{R}$.

1) For the first data $k = 1$, $M = 1$, $v_1(1) = y(1)$, ($i = 1 \dots N$), $c_{i1}(1) = x_i(1)$, $\sigma_{i1}(1) = rand \in (0, 1)$.

2) For the other data $k \geq 2$, evaluate $z_j(k-1)$ and $b(k-1)$ with (2), evaluate $\hat{y}(k-1)$ with (2), evaluate $p(k-1)$ with (3), update $c_{ij}(k)$, $\sigma_{ij}(k)$ and $v_j(k)$ with (7), (8) and (9). where $D_{1ij}(k-1)$, $D_{2ij}(k-1)$, and $D_{3j}(k-1)$ are given in (7).

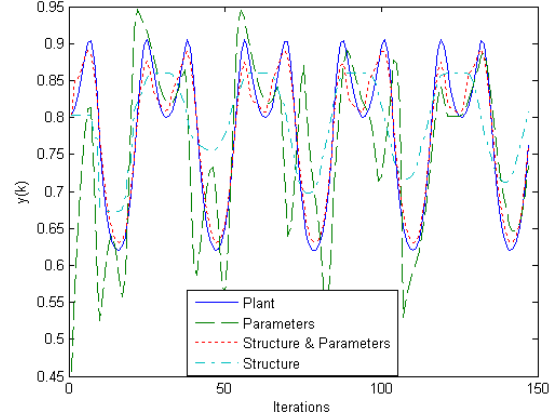


Figure 1. Identification results for the example 1

3) if $p(k-1) < r$, then a new rule is generated ($M = M+1$) where $r \in (0, 1)$, assign initial values to $c_{i,M+1}(k)$, $\sigma_{i,M+1}(k)$, and $v_{M+1}(k)$ with (4), go to 2.

4) If $p(k-1) \geq r$, then a rule is not generated and in the case that $z_j(k-1) = p(k-1)$ we have the winner rule j^* , this rule is updated with (5), go to 2.

6 Simulation

In this section, the suggested online evolving proposed algorithm is applied to nonlinear system identification. In both examples the least mean square error is used for the comparison results. The least mean square error is defined as [9]:

$$MSE = \frac{1}{N} \sum_{k=1}^N e^2(k-1) \quad (10)$$

Example 1 Consider the nonlinear system given in [14]:

$$y(k) = 0.52 + \frac{0.1x_1 + 0.28x_2}{x_1^2 + x_2^2} - 0.06x_1x_2 \quad (11)$$

with $x_1(k) = \sin^2(10/k)$ and $x_2(k) = \cos^2(10/k)$. We compare our algorithm called Structure & parameters using the parameters $\eta = 0.9$, $r = 0.6$ with the case where only the structure is trained called Structure using the parameter $r = 0.6$ and with the case where the only the parameters are trained called Parameters using the parameter $\eta = 0.3$ and 2 neurons in the hidden layer. The comparison results of the identification are given in Figure 1 and in Table 1.

Table 1: Results for Example 1

Methods	Rules	MSE
Structure	4	0.0023
Parameters	2	0.0045
Structure & parameters	2	1.6409×10^{-4}

The least mean square error for the Structure algorithm is 0.0023, for Parameters algorithm is 0.0045 and for Structure & parameters

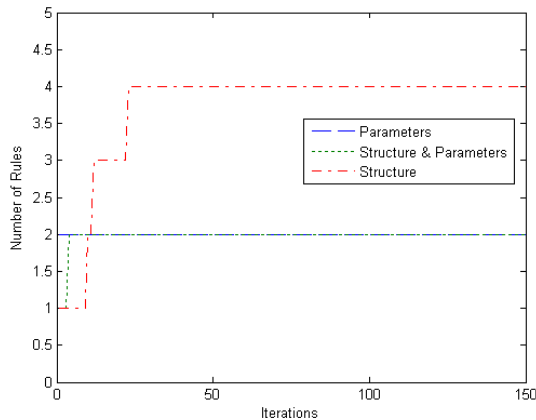


Figure 2. Number of rules for example 1

algorithm is 1.6409×10^{-4} . The Structure algorithm generate 4 rules and the Structure & parameters algorithm generate 2 rules. In Figure 2, the comparison results for the number of rules is shown.

Form the simulation results shown in example 1 in it can be seen that the proposed algorithm has a good behavior for the identification of nonlinear systems.

7 Conclusion

In this paper we presented a quick and efficient approach for system modeling using the the backpropagation algorithm to train an evolving radial basis function network. The proposed neural fuzzy network uses the online clustering to train the structure, the back-propagation to train the parameters. The structure identification and parameter learning are done online.

REFERENCES

- [1] M.F. Azem, M.Hanmandlu and N. Ahmad, Structure identification of generalized adaptive neuro-fuzzy inference systems, *IEEE Trans. on Fuzzy Syst.*, Vol.11, No.6, 666-681, 2003.
- [2] M. Brown, C.J. Harris, *Adaptive Modelling and Control*, Macmillan Pub.Co., Prentice Hall, New York, 1994.
- [3] S.L. Chiu, Fuzzy Model Identification based on cluster estimation, *Journal of Intelligent and Fuzzy Systems*, Vol.2, No.3, 1994.
- [4] J.R. Hilera, V.J. Martinez, *Redes Neuronales Artificiales, Fundamentos, Modelos y Aplicaciones*, Adison Wesley Iberoamericana, U.S.A , 1995.
- [5] J. S. R. Jang, AFNFIS: Adaptive-Network-Based Fuzzy inference system, *IEEE Trans. on Systems, Man and Cybernetics*, Vol.23, 665-685, 1993.
- [6] J. S. R. Jang and C. T. Sun, *Neuro-Fuzzy and Soft Computing*, Prentice Hall, NJ 07458, 1997.
- [7] C.F. Juang and C.T. Lin, An On-line Self Constructing Nural Fuzzy Inference Network and Its Applications, *IEEE Trans. on Fuzzy Syst.*, Vol.6, No.1, 12-32, 1998.
- [8] C.F. Juang and C.T. Lin, A Recurrent Self-Organizing Fuzzy Inference Network, *IEEE Trans. on Neural Networks*, Vol.10, No.4, 828-845, 1999.
- [9] N. Kasabov, Evolving Fuzzy Nural Networks for Supervised/Unsupervised Online Knowledge-Based Learning, *IEEE Trans. on Systems, Man and Cybernetics*, Vol.31, No.6, 902-918, 2001.
- [10] C.T.Lin, *Neural Fuzzy Control Systems with Structure and parameter learning*, New York: World Scientific, 1994.

- [11] S. Mitra, Y. Hayashi, Neuro-Fuzzy rule Generation: Survey in Soft Computing Framework, *IEEE Trans. on Neural Networks*, Vol.11, No.3, 748-769, 2000.
- [12] I. Rivals and L. Personnaz, Neural Network Construction and Selection in non linear modelling, *IEEE Trans. on Neural Networks*, Vol.14, No.4, 804-820, 2003.
- [13] S.G.Tzafestas and K.C. Zikidis, On-Line Neuro-Fuzzy ART-Based Structure and parameter learning TSK Model, *IEEE Trans. on Systems, Man and Cybernetics*, Vol.31, No.5, 797-803, 2001.
- [14] L. X. Wang, *A Course in Fuzzy Systems and Control*, Prentice Hall, Englewood Cliffs, NJ 07458, 1997.

Cascaded Multiresolution Spline-based Fuzzy Neural Network

Vitaliy Kolodyazhnyi* and Yevgeniy Bodyanskiy, *Senior Member, IEEE*

Abstract—A novel cascaded multiresolution spline-based fuzzy neural network (CMS-FNN) with a very fast constructive training algorithm is introduced. Structurally the CMS-FNN resembles the known cascade-correlation architecture but differs from it in the type of neurons as well as in the training algorithm. The neurons in the CMS-FNN are a generalization of the so-called neo-fuzzy neurons (NFNs) via substitution of B-splines for triangular membership function (MF) while the training algorithm relies on direct linear least squares estimation of the synaptic weights due to the linearity of the NFNs in their parameters. Training of the CMS-FNN requires only as many epochs as the number of neurons created in the network in the process of network construction. The number of MFs of the NFNs in the CMS-FNN model is increased from layer to layer to form a multi-resolution model. The efficiency of the proposed model is confirmed by long-range iterated predictions of three chaotic time series. An adaptive modification of the proposed approach is discussed and is quite straightforward.

I. INTRODUCTION

ARTIFICIAL neural networks [3] and hybrid fuzzy neural networks (FNNs) [4] have been attracting increasingly more and more attention of researchers since the early 90s. Indeed, the computational intelligence approach to nonlinear system modeling, prediction, and classification proved to be very efficient and found numerous applications. Dozens of architectures and training algorithms were proposed, including the evolving models that can operate with little prior information about the modeled system and can determine their own structure in real time and adapt their parameters to track changes [1].

At the same time, the problem of slow convergence of training algorithms relying on error backpropagation and the curse of dimensionality for systems with a large number of input variables still remain important. To tackle these problems, we previously proposed the neuro-fuzzy Kolmogorov's network (NFKN) architecture [5]–[8] based on the so called neo-fuzzy neurons (NFNs) [12] which have many remarkable properties, one of them being the linearity of the neuron output in its synaptic weights. These properties helped us accelerate the training of the NFKN

compared to conventional architectures maintaining high precision, and also develop deterministic initialization procedures [5]–[8] which make unnecessary the usual repetitive training of the conventional neural networks [1] using random initial weights. However, the training of the NFKN still requires from one to several hundreds of training epochs depending on the problem [5], [6], [8].

In this paper we make an attempt to develop a novel neuro-fuzzy architecture that does not require backpropagation for its training, can be trained in several epochs, provides high precision of nonlinear system modeling, and is scalable for problems with a large number of inputs.

To construct the new architecture, we also use the NFNs as in the NFKN model [5]–[7] but in a different cascaded architecture. This enables us to derive a very fast derivative-free constructive training algorithm based on linear least squares. To improve the approximation accuracy of the cascaded neuro-fuzzy network, we generalize the NFNs by using B-spline functions as in our previous work [8].

II. ARCHITECTURE

The proposed cascaded multiresolution spline-based fuzzy neural network (CMS-FNN) is shown below in Fig. 1.

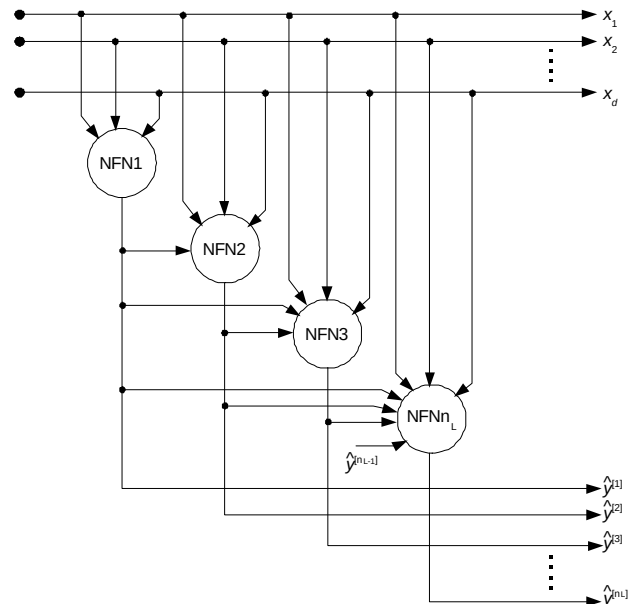


Fig. 1. Cascaded FNN

Structurally, the architecture of the CMS-FNN resembles that of a cascade-correlation neural network [2]. However,

*This research was in part supported by the 6th Framework Project EUCLOCK (No. 018741).

Vitaliy Kolodyazhnyi is with the Institute for Psychology, University of Basel, Missionsstrasse 60/62, 4055 Basel, Switzerland. Tel.: +41 61 267 03 83, fax: +41 61 267 06 48, e-mail: v.kolodyazhnyi@unibas.ch.

Yevgeniy Bodyanskiy is head of the Control Systems Research Laboratory at Kharkiv National University of Radioelectronics, 14, Lenin av., 61166, Kharkiv, Ukraine. Tel: +38 057 702 18 90, e-mail: bodya@kture.kharkov.ua.

we employ NFNs [12] instead of conventional artificial neurons with a sigmoid activation function as in [2].

The number of neurons in a layer is equal to the number of output variables in the modeled system, e.g. there is one neuron in each layer if there is only one output variable. Without loss of generality, we will further consider modeling of nonlinear systems with one output variable. Generalization to multiple output variables is trivial.

The use of NFNs, as will be shown below, enabled us to develop a new very fast constructive training algorithm. The structure of an NFN and a conventional neuron is shown in Fig. 2.

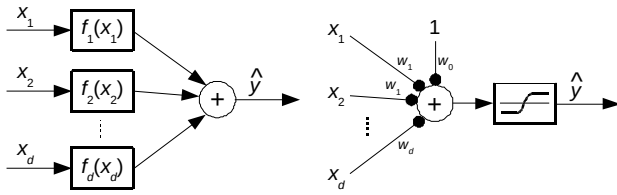


Fig. 2. Neo-fuzzy neuron and a conventional artificial neuron

The output of an NFN is computed according to the following equations [12]:

$$\hat{y} = \sum_{i=1}^d f_i(x_i), \quad f_i(x_i) = \sum_{h=1}^m \mu_{i,h}(x_i) w_{i,h}, \quad i=1, \dots, d, \quad (1)$$

where $\hat{y}$ is the neuron output, $f_i(x_i)$ is the function of the i -th nonlinear synapse, x_i is the i -th input, d is the number of inputs, $\mu_{i,h}$ is the h -th membership function (MF) on the input i , $w_{i,h}$ is the weight of the MF $\mu_{i,h}$, and m is the number of MFs per input. Alternatively, equations (1) can be represented in form of fuzzy rules [5]-[8]:

$$\text{IF } x_i \text{ IS } X_{i,h} \text{ THEN } \hat{y} = w_{i,h} d, \quad i=1, \dots, d, \quad h=1, \dots, m, \quad (2)$$

where $X_{i,h}$ is the linguistic term defined by $\mu_{i,h}$.

The CMS-FNN architecture consists of n_L layers (see Fig. 1). The output of layer l is computed according to equation

$$\hat{y}^{[l]} = \begin{cases} \sum_{i=1}^d \sum_{h=1}^{m_i} \mu_{i,h}^{[1]}(x_i) w_{i,h}^{[1]}, & \text{if } l=1, \\ \sum_{i=1}^d \sum_{h=1}^{m_i} \mu_{i,h}^{[l]}(x_i) w_{i,h}^{[l]} + \sum_{j=1}^{l-1} \sum_{h=1}^{m_j} \mu_{j+d,h}^{[l]}(\hat{y}^{[j]}) w_{j+d,h}^{[l]}, & \text{if } l=2, \dots, n_L, \end{cases} \quad (3)$$

where m_l is the number of membership functions per synapse in the NFN of the layer l . We choose these numbers as $m_l = m_{l-1} + 1$ for $l=2, \dots, n_L$. In such a way, multi-resolution fuzzy partitions are induced where only the number of MFs in the first layer given by m_1 needs to be specified.

As is suggested in works [13] and [8], it is advantageous to define the MFs in a fuzzy system as B-splines (“basic splines”). Given a sequence of r ordered knots $\{c_1, \dots, c_r\}$, the p -th B-spline function of order q is defined as

$$N_{p,q}(x) = \begin{cases} 1, & \text{for } c_p \leq x < c_{p+1}, \\ 0, & \text{otherwise,} \end{cases} \quad \text{if } q=1, \quad (4)$$

$$N_{p,q}(x) = \begin{cases} \frac{x-c_p}{c_{p+q-1}-c_p} N_{p,q-1}(x) \\ + \frac{c_{p+q}-x}{c_{p+q}-c_{p+1}} N_{p+1,q-1}(x), & \text{if } q > 1, \end{cases}$$

$$p=1, \dots, r-q.$$

Remarkably, one obtains triangular MFs letting $q=2$ in (4). Below we will consider B-splines for $q=2$ and $q=4$, the latter case being cubic functions. For practical use, also quadratic B-splines for $q=3$ can be of interest. For a spline-based neo-fuzzy neuron (1), the MFs are defined as $\mu_{i,h}(x_i) = N_{h,q}(x_i)$. As in the NFKN model [5]-[8], we assume that the MFs are fixed and equidistantly spaced over the range of each NFN input. The parameters of the MFs (spline knots) are not tuned.

To maintain the partition of unity throughout the universe of discourse of x , marginal functions are required at both ends of the universe of discourse of x in (4) for $q > 2$ (see Fig. 3) [8], [13]. The minimum number of B-functions per input is two for $q=2$ and four for $q=4$ including the marginal functions.

To further guarantee the partition of unity also for inputs outside the universe of discourse of x , we set the leftmost and rightmost knots to very large negative and positive values, respectively [8].

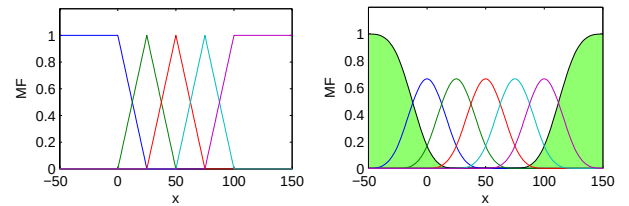


Fig. 3. B-spline membership functions of order 2 (left) and 4 (right) defined for variable $x \in [0, 100]$ such that 5 membership function are defined over the universe of discourse of x . Shaded areas (right) correspond to marginal B-functions

It should be noted that for B-spline MFs their normality is guaranteed only for $q \leq 2$, i.e. for higher orders the maximum of an MF is generally below one. This can be seen from comparison of MFs for $q=2$ and $q=4$ in Fig. 3. For $q \leq 2$, the maximum of an MF depends on the location of its knots. Setting the leftmost and rightmost knots for the marginal MFs for $q > 2$ to very large negative and positive values, respectively, results in maxima of the marginal functions close to one.

III. TRAINING ALGORITHM

The weights of the CMS-FNN are determined by means of a batch-training algorithm briefly outlined below.

Let us assume that the training of the network starts from the first layer, i.e. from one single NFN receiving $x_1, \dots, x_d$ as its inputs. At this point no further layers exist. Since the nonlinear synapses in (1) are linear in their parameters, the synaptic weights of the first layer can be estimated with direct linear least squares (LS) optimization.

After determining the synaptic weights for the first layer, we “freeze” them and compute $\hat{y}^{[1]}$ according to (3). Then we can add a second layer and again use the LS procedure for determining the weights of the second layer.

The same procedure can be repeated to construct n_L layers. A stopping criterion can be based, e.g., on accuracy or a maximum number of weights in the network.

For each layer $l = 1, \dots, n_L$ the minimized error function is

$$E^{[l]} = \frac{1}{2} \sum_{k=1}^K [y(k) - \hat{y}^{[l]}(k)]^2 = \frac{1}{2} [Y - \hat{Y}^{[l]}]^T [Y - \hat{Y}^{[l]}], \quad (5)$$

where $Y = [y(1), \dots, y(K)]^T$ is the vector of target values, and $\hat{Y}^{[l]} = [\hat{y}^{[l]}(1), \dots, \hat{y}^{[l]}(K)]^T$ is the vector of network outputs, and K is the number of data points in the training data set. To obtain the LS solution for the l -th layer, rewrite (3) as

$$y^{[l]} = \begin{cases} W^{[1]T} \phi^{[1]}(x_1, \dots, x_d), & \text{if } l = 1, \\ W^{[l]T} \phi^{[l]}(x_1, \dots, x_d, \hat{y}^{[1]}, \dots, \hat{y}^{[l-1]}), & \text{if } l = 2, \dots, n_L. \end{cases} \quad (6)$$

We find a regularized LS solution as

$$W^{[l]} = \left(\Phi^{[l]T} \Phi^{[l]} + \eta_l I \right)^{-1} \Phi^{[l]T} Y, \quad l = 1, \dots, n_L, \quad (7)$$

where for the first layer $W^{[1]} = [w_{1,1}^{[1]}, w_{1,2}^{[1]}, \dots, w_{d,m_1}^{[1]}]^T$,

$\Phi^{[1]} = [\phi^{[1]}(x_1(1), \dots, x_d(1)), \dots, \phi^{[1]}(x_1(K), \dots, x_d(K))]^T$,
 $\phi^{[1]}(x_1, \dots, x_d) = [\mu_{1,1}^{[1]}(x_1), \mu_{1,2}^{[1]}(x_1), \dots, \mu_{d,m_1}^{[1]}(x_d)]^T$, and for the other layers for $l = 2, \dots, n_L$:

$$\begin{aligned} W^{[l]} &= [w_{1,1}^{[l]}, w_{1,2}^{[l]}, \dots, w_{d,m_l}^{[l]}, w_{d+1,1}^{[l]}, w_{d+1,2}^{[l]}, \dots, w_{d+l-1,m_l}^{[l]}]^T, \\ \Phi^{[l]} &= [\phi^{[l]}(x_1(1), \dots, x_d(1), \hat{y}^{[1]}, \dots, \hat{y}^{[l-1]}), \dots, \\ &\dots, \phi^{[l]}(x_1(K), \dots, x_d(K), \hat{y}^{[1]}, \dots, \hat{y}^{[l-1]})]^T, \\ \phi^{[l]}(x_1, \dots, x_d, \hat{y}^{[1]}, \dots, \hat{y}^{[l-1]}) &= \\ &[\mu_{1,1}^{[l]}(x_1), \mu_{1,2}^{[l]}(x_1), \dots, \mu_{d,m_l}^{[l]}(x_d), \mu_{d+1,1}^{[l]}(\hat{y}^{[1]}), \mu_{d+1,2}^{[l]}(\hat{y}^{[1]}), \dots, \\ &\dots, \mu_{d+l-1,m_l}^{[l]}(\hat{y}^{[l-1]})]^T. \end{aligned}$$

In the CMS-FNN training algorithm the number of training epochs is equal to the number of layers n_L that are created in the network. It is expected that for most practical applications no more than ten layers will be required.

The parameter η_l is an adjustable regularization term used to prevent rank deficiency because in the general case the matrix $\Phi^{[l]T} \Phi^{[l]}$ cannot be inverted directly. As in [8], we compute the parameter η_0 as follows:

$$\eta_l = \eta_0 \max(\text{eig}(\Phi^{[l]T} \Phi^{[l]})), \quad l = 1, \dots, n_L, \quad (8)$$

i.e. η_l is proportional to the largest eigenvalue of the matrix $\Phi^{[l]T} \Phi^{[l]}$ and η_0 is a small constant usually chosen in the range $10^{-9} \dots 10^{-3}$ which controls the smoothness of the obtained solution. This is also important for noisy data.

IV. EXPERIMENTS

In our experiments we tested the performance of models with membership functions of order $q = 2$ (triangular MFs) and $q = 4$ (cubic B-spline MFs) in iterated predictions of the Mackey-Glass (MG) time series [9] generated for $\tau = 17$ and $\tau = 30$, and the laser time series from Santa-Fe time series competition [11].

For the MG time series, we trained networks with 17 and 30 delayed inputs using 3000 data points from 118 to 3117 and 131 to 3130 to predict the time series one step ahead. After training, the networks were used for iterated predictions of the next 1000 points without ‘knowing’ the actual data of the time series for these 1000 points.

For the laser time series, networks with 50 delayed inputs were trained using 950 data points from 51 to 1000, and the iterative predictions were performed for the next 100 points from 1001 to 1100.

For all the three time series, the best performance was achieved with $q = 4$ (cubic B-spline MFs). In all of the tested networks with both types of MFs we set $m_1 = 4$. For the MG data with $\tau = 17$ and the laser data the best prediction was achieved with seven layers, and with six layers for the MG data with $\tau = 30$. The best results are shown in Fig. 4-6.

For both of the MG time series the real process and its prediction remain nearly indistinguishable for 1000 time steps and the difference can be seen only in the error curves with a finer scale.

For the laser data, the iterated prediction is somewhat worse than that in [8] and [11]. However, the CMS-FNN model was able to correctly predict the abrupt change in the amplitude of intensity oscillations after the 1060th time step that is a very difficult prediction task per se. For the MG time series prediction our results rank among the very best achieved so far.

The CMS-FNN model and its training were implemented in MATLAB 6.5. In all cases the training time did not exceed 6 seconds on a PC with an Intel Core2 Duo CPU running at 2 GHz with 2 GB of RAM.

V. CONCLUSION

A new cascaded neuro-fuzzy architecture with a very fast derivative-free constructive training algorithm was proposed. Although the CMS-FNN model is structurally similar to a cascade correlation neural network [2], the training of CMS-FNN is much faster as it is based on direct linear least squares estimation and requires only as many epochs as the number of layers.

The constructive training of the CMS-FNN model is also similar to that of the GMDH neural nets [10] but is much computationally simpler and is expected to be better scalable for problems with a large number of input variables.

An adaptive modification of the CMF-FNN training algorithm for online updating of weights and possibly structure is straightforward and will be implemented by the authors in the nearest future. Such an implementation would include a recursive least squares procedure and an online multi-step algorithm for updating the synaptic weights from layer to layer. New layers can be added or existing layers discarded without affecting the synaptic weights of the underlying layers due to the nature of the CMF-FNN model.

REFERENCES

- [1] P. Angelov, *Evolving Rule-Based Models*, Heidelberg-New York: Physica-Verlag, 2002.
- [2] S.E. Fahlman, C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems*, D.S. Touretzky, Ed. San Mateo, CA: Morgan Kaufman, 1990, pp. 524-532.
- [3] S. Haykin, *Neural networks: A comprehensive foundation*. Upper Saddle River: Prentice Hall, 1999.
- [4] J.-S. R. Jang, C.-T. Sun, and E. Mizutani, *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Upper Saddle River: Prentice Hall, 1997.
- [5] V. Kolodyazhniy and Ye. Bodyanskiy, "Fuzzy Kolmogorov's Network," in *Lecture Notes in Computer Science* vol. 3214, M.G. Negoita et al., Ed. Berlin: Springer-Verlag, 2004, pp. 764-771.
- [6] V. Kolodyazhniy, Ye. Bodyanskiy, and P. Otto, "Universal Approximator Employing Neo-Fuzzy Neurons," in *Computational Intelligence, Theory and Applications. Advances in Soft Computing*, vol. 33, B. Reusch, Ed. Berlin: Springer-Verlag, 2005, pp. 631-640.
- [7] V. Kolodyazhniy, F. Klawonn, and K. Tschumitschew, "A Neuro-Fuzzy Model for Dimensionality Reduction and Its Application," *International Journal of Uncertainty, Fuzziness, and Knowledge-Based Systems*, vol. 15, 2007, pp. 571-593.
- [8] V. Kolodyazhniy, "Spline-based neuro-fuzzy Kolmogorov's network for time series prediction," in *Proc. 17th European Symposium on Artificial Neural Networks*, Bruges, Belgium, 2009, pp. 153-158.
- [9] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, 1977, pp. 287-289.
- [10] D. T. Pham and X. Liu, *Neural Networks for Identification, Prediction and Control*, London: Springer-Verlag, 1995.
- [11] A. S. Weigend and N. A. Gershenfeld, "Results of the time series prediction competition at the Santa Fe Institute," in *Proc. IEEE Int. Conf. on Neural Networks*, San Francisco, CA, 1993, vol. 3, pp. 1786-1793.
- [12] T. Yamakawa, E. Uchino, T. Miki, and H. Kusanagi, "A neo fuzzy neuron and its applications to system identification and prediction of the system behavior," in *Proc. 2nd Int. Conf. on Fuzzy Logic and Neural Networks (IIZUKA-92)*, Iizuka, Japan, 1992, pp. 477-483.
- [13] J. Zhang and A. Knoll, "Constructing Fuzzy Controllers with B-Spline Models— Principles and Applications," *Int. J. of Intelligent Systems*, vol. 13, 1998, pp. 257-285.

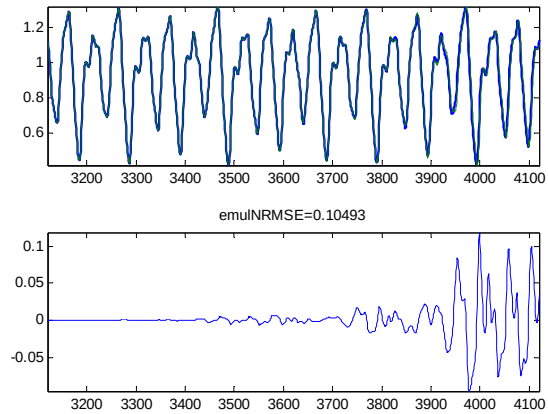


Fig. 4. MG time series for $\tau = 17$ (thick line in the upper plot), iterated prediction (thin line in the upper plot), and prediction error (lower plot)

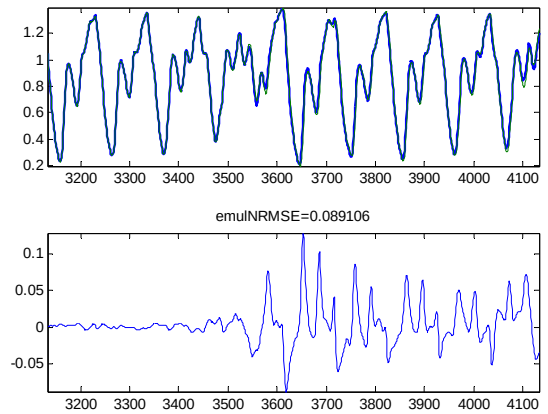


Fig. 5. MG time series for $\tau = 30$ (thick line in the upper plot), iterated prediction (thin line in the upper plot), and prediction error (lower plot)

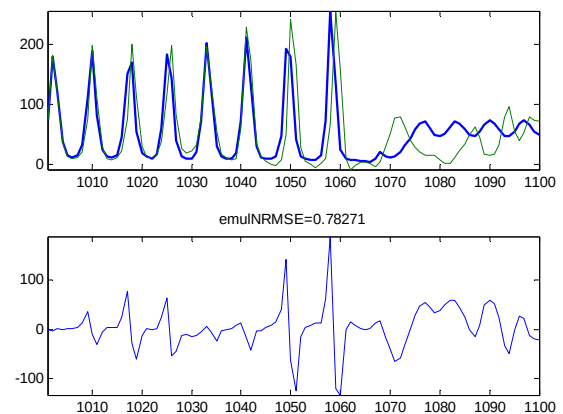


Fig. 6. Laser time series (thick line in the upper plot), iterated prediction (thin line in the upper plot), and prediction error (lower plot)

The Need for Benchmarks with Data from Stochastic Processes and Meta-Models in Evolving Systems

Katharina Tschumitschew
Department of Computer Science
Ostfalia University of Applied Sciences
Salzdahlumer Str. 45/48
D-38302 Wolfenbuettel, Germany
Email: katharina.tschumitschew@ostfalia.de

Frank Klawonn
Department of Computer Science
Ostfalia University of Applied Sciences
Salzdahlumer Str. 45/48
D-38302 Wolfenbuettel, Germany
Email: f.klawonn@ostfalia.de
and

Bioinformatics and Statistics
Helmholtz Centre for Infection Research
Inhoffenstrasse 7
D-38124 Braunschweig, Germany
Email: frank.klawonn@helmholtz-hzi.de

Abstract—The problem with real data is often that noise and forms of randomness are involved. This causes a specific problem for evolving systems. They must be able to distinguish between changes according to noise and changes of the underlying data generating process or its parameters. In the worst case, an evolving system might just try to track the noise in the data and is unable to learn the actual relationship inherent in the data. But how can we make sure that this will not be the case for a given evolving system? We propose some simple benchmark tests in our paper that can give an idea of how much an evolving system might be misled by noise.

I. INTRODUCTION

Evolving systems are designed to cope with streaming data. There are two important aspects that are usually considered for evolving systems. It is required that the evolving system can react in real time so that efficient computations – usually recursive ones – are needed without analyzing the whole data that have been collected so far again. The second aspect is the change of the underlying process that generates the data, so that the evolving system must adapt itself on-line.

The problem with real data is often that noise and forms of randomness are involved. This causes a specific problem for evolving systems. They must be able to distinguish between changes according to noise and changes of the underlying data generating process or its parameters.

Statisticians categorize models that do not make explicit assumptions about the data generating process as exploratory data analysis techniques. Such methods, like decision or regression trees, neural networks or support vector machines are very successful in classification and regression tasks, but there is a need to evaluate the performance of such models, since they cannot be analyzed in the classical statistical sense where the underlying assumptions about the data generating process

must be made explicit. A very common way to evaluate the performance of such models are methods that divide the data into training and test data like, for instance 10-fold cross-validation. The training data are used to construct the model and the performance is evaluated based on the test data.

For evolving systems, the partition into training and test data is more complicated. We cannot simply take out some data for testing, since the data generating process is assumed to change and the evolving system is intended to track these changes. What could be done is to take out single data records of a time series and see how the evolving system performs with respect to this records. However, to be representative, we would need a much larger test data set due to the fact that the data generating process changes and the test data must reflect situations with changes and without changes accordingly. Even then it is not clear how to judge the performance of an evolving system. A large error can be due to overfitting – i.e. erroneously tracking the noise in the data – or because there is so much noise in the data that the performance cannot be better. A rough indication of how much an evolving system tries to track the noise, is the difference between the error on the training and the test data. However, if an evolving system has already a bad performance on the training data, the performance on the test data cannot be much worse and one might assume that the error comes from the noise in the data.

In this paper, we propose to set up simple theoretical models for the data generating process for which we can give at least a rough answer to the question, how well a model that takes assumptions on the data generating process into account could perform. We can then compare how much worse the evolving system performs. Of course, we cannot expect the evolving system to have a similar performance, since it is not allowed to make specific assumptions about the data generating process. But this comparison will give us an idea, how much an evolving system can deviate from an optimal model and whether a simpler model that would not track the noise might not have a better performance.

We consider simple prediction tasks like classification and regression in this paper. Evolving system have been proposed for such problems for instance in [1], [2], [4], [5], [6].

In section II we carry out a theoretical analysis between an extremely simplified evolving system based on a windowing technique and a model tailored to the known assumptions of the data generating process. This can be considered as a regression problem with the identity function as the regression function.

Section III introduces a simple switching model which can be interpreted as a regression or a classification problem. Here we carry out an experimental comparison between a maximum likelihood estimator exploiting the assumptions on the underlying data generating process and an evolving system without specific assumptions on the data generating process.

We conclude our paper with remarks on more complicated prediction tasks.

II. RANDOM WALK

A very simple example for systems with a random change over time is one dimensional random walks [7]. A random

walk $(X_t)_{t \in \mathbb{N}}$ is obtained by adding up values from independent and identically-distributed (i.i.d.) random variables Y_i with expected value zero, i.e. $E(Y_i) = 0$ and variance $\sigma^2 = \text{Var}(Y_i) = \text{Var}(Y)$.

$$X_t = \sum_{i=1}^t Y_i \quad (1)$$

The expected value for the random walk is

$$E(X_t) = E\left(\sum_{i=1}^t Y_i\right) = \sum_{i=1}^t E(Y_i) = 0 \quad (2)$$

independent of t , whereas the variance of random walk increases linearly with t .

$$\text{Var}(X_t) = \text{Var}\left(\sum_{i=1}^t Y_i\right) = t \cdot \text{Var}(Y). \quad (3)$$

The covariance of a random walk is given by

$$\text{Cov}(X_t, X_s) = s \cdot \text{Var}(Y), \quad (s < t) \quad (4)$$

and tends to infinity as well with increasing difference between the time points t and s .

The best prediction that one can do for a random walk is the naïve approach, simply using the last value is a prediction for the next value. $\hat{x}_{t+1} = x_t$. Therefore, the difference between the true value and the predicted value is $X_{t+1} - X_t = Y_{t+1}$. Thus, the expected quadratic error is

$$E((X_{t+1} - X_t)^2) = E(Y_{t+1}^2) = \text{Var}(Y). \quad (5)$$

Now assume, we do not know that we deal with a random walk and try an extremely simple “evolving system” using a windowing technique with a window size of T , using the mean of the last T values as a prediction for the next value. The expected quadratic error for the prediction can then be computed as follows, where $t_0 = t - T + 1$ is the start of the window:

$$\begin{aligned} E\left(\left(X_{t+1} - \frac{1}{T} \sum_{i=t_0}^t X_i\right)^2\right) &= \\ E\left(X_{t+1}^2 - \frac{2}{T} \sum_{i=t_0}^t X_i \cdot X_{t+1} + \frac{1}{T^2} \left(\sum_{i=t_0}^t X_i\right)^2\right) &= \\ E(X_{t+1}^2) - \frac{2}{T} \sum_{i=t_0}^t E(X_i \cdot X_{t+1}) + \frac{1}{T^2} E\left(\left(\sum_{i=t_0}^t X_i\right)^2\right) &= \\ \text{Var}(X_{t+1}) - \frac{2}{T} \sum_{i=t_0}^t \text{Cov}(X_i, X_{t+1}) + \\ \frac{1}{T^2} E\left(\sum_{i=t_0}^t X_i^2 + 2 \sum_{i=t_0+1}^t \sum_{j=t_0}^{i-1} X_i \cdot X_j\right) &= \\ \text{Var}(X_{t+1}) - \frac{2}{T} \sum_{i=t_0}^t \text{Cov}(X_i, X_{t+1}) + \\ \frac{1}{T^2} \left(\sum_{i=t_0}^t \text{Var}(X_i) + 2 \sum_{i=t_0+1}^t \sum_{j=t_0}^{i-1} \text{Cov}(X_i, X_j)\right) &= \\ (t+1) \cdot \text{Var}(Y) - \frac{2}{T} \sum_{i=t_0}^t i \cdot \text{Var}(Y) + \\ \frac{1}{T^2} \left(\sum_{i=t_0}^t i \cdot \text{Var}(Y) + 2 \sum_{i=t_0+1}^t \sum_{j=t_0}^{i-1} j \cdot \text{Var}(Y)\right) &= \end{aligned}$$

$$\begin{aligned} \text{Var}(Y) \left(t+1 - \frac{2}{T} \sum_{i=t_0}^t i + \frac{1}{T^2} \sum_{i=t_0}^t i + 2 \sum_{i=t_0+1}^t \sum_{j=t_0}^{i-1} j \right) &= \\ \frac{(2t - 2t_0 + 3)(t - t_0 + 2)}{6(t - t_0 + 1)} \text{Var}(Y) &= \end{aligned} \quad (6)$$

It is easy to show that (6) has its minimum at $t_0 = t$, i.e. for a window of size one, where we simply use the last value as a prediction for the next value. The worst case is to use all values, i.e. $t_0 = 0$. In this case, the expected quadratic error tends to infinity with increasing t . Figure 1 shows (6) as a function in t for $t_0 = 0$ and $\text{Var}(Y) = 1$. In this case, t can be interpreted as the window size. The expected quadratic error increases with $O(T)$ in terms of the window size T .

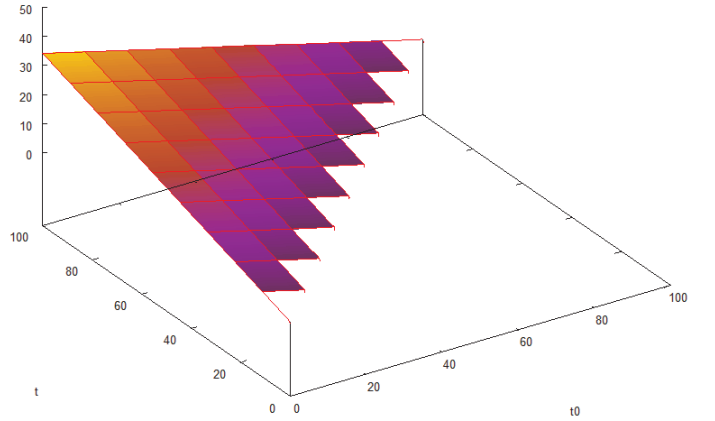


Fig. 1. The expected quadratic error for a random walk prediction depending on the window size.

One might criticize that the mean over a window is a much too simple prediction. But actually, any more complicated function will tend to make the expected quadratic error even worse.

The next example is a random walk to which we add noise in the form of a random variable Z with $E(Z) = 0$.

$$X_t = \sum_{i=1}^t Y_i + Z_t \quad (7)$$

where the Y_i are i.i.d. as Y and the Z_t are i.i.d. as Z . We assume also that the random variables Y_i and Z_j are all independent. Since $E(Y) = 0$ and $E(Z) = 0$ holds, we have again $E(X_t) = 0$.

As in the previous example of the random walk without noise, the best prediction for the next value we can choose is the previous value: $\hat{x}_{t+1} = x_t$. The difference between the true and the predicted value is then $X_{t+1} - X_t = Y_{t+1} + Z_{t+1} - Z_t$.

The variance and covariance of a random walk with noise are

$$\text{Var}(X_t) = t \cdot \text{Var}(Y) + \text{Var}(Z) \quad (8)$$

and

$$\text{Cov}(X_t, X_s) = s \cdot \text{Var}(Y), \quad (s < t), \quad (9)$$

respectively.

The expected quadratic error can be computed as follows:

$$\begin{aligned} E\left((X_{t+1} - X_t)^2\right) &= E\left((Y_{t+1} + Z_{t+1} - Z_t)^2\right) \\ &= \text{Var}(Y) + 2\text{Var}(Z) \end{aligned} \quad (10)$$

Based on (8) and (9) we can calculate the expected quadratic error for the prediction based on a windowing technique.

$$\begin{aligned} E\left(\left(X_{t+1} - \frac{1}{T} \sum_{i=t_0}^t X_i\right)^2\right) &= \\ E\left(X_{t+1}^2 - \frac{2}{T} \sum_{i=t_0}^t X_i \cdot X_{t+1} + \frac{1}{T^2} \left(\sum_{i=t_0}^t X_i\right)^2\right) &= \\ E(X_{t+1}^2) - \frac{2}{T} \sum_{i=t_0}^t E(X_i \cdot X_{t+1}) + \frac{1}{T^2} E\left(\left(\sum_{i=t_0}^t X_i\right)^2\right) &= \\ \frac{(2t - 2t_0 + 3)(t - t_0 + 2)}{6(t - t_0 + 1)} \text{Var}(Y) + 2(t - t_0 + 1) \text{Var}(Z) \end{aligned} \quad (11)$$

As before, this function has its minimum at $t_0 = t$ (window size $T = 1$) and is largest for the highest window size. The expected quadratic error is also increasing in $O(T)$.

III. SWITCH MODEL

The two random walk examples described in the previous section can be understood as data generating models that change continuously, representing very simple examples for shift. In this section, we discuss an example for a switching model with sudden jumps in the change of the data generating process. To keep the model as simple as possible, we consider a data generating process that switches between two normal distributions with the same variance σ^2 , but one with expected value $\mu_1 = 0$ and the other with expected value $\mu_2 = 1$. The probability to switch from one normal distribution to the other is p in each step and the probability to stay with the same normal distribution is $(1 - p)$. The random switching between the two normal distributions is carried out independently in each time step. This model for the data generating process is illustrated as an automaton with two states in figure 2.

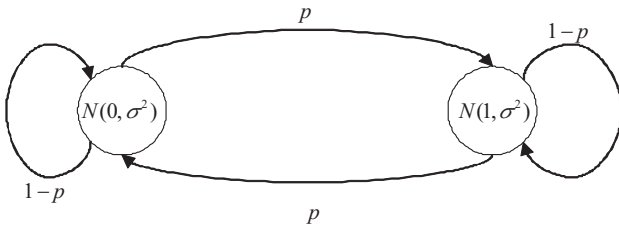


Fig. 2. A switching model.

Depending on the switching probability p , we can distinguish three cases.

- $p \approx \frac{1}{2}$ The data are generated randomly from any of the two normal distributions in each step of the process. It is impossible to make a prediction from which normal distribution the next value will be drawn based on previous data.

- $p \gg \frac{1}{2}$ There is tendency to permanently switch from one normal distribution to the other in each step.
- $p \ll \frac{1}{2}$ The data generating process tends to stay with the same normal distribution in each step and switches only once in a while.

The first case does not really represent a switching model. It can be understood as drawing independent random samples from a mixture of the two normal distributions. The second case is less interesting from the practical point of view. We normally assume that normally the model will not stay stable and changes happen only once in a while. Therefore, we only consider the last case. There it is interesting to consider the relation between p and σ^2 . Small values for p , i.e. changing from one normal distribution to the other rarely happens, and small values for σ^2 , i.e. we distinguish between values from the one or the other normal distribution with a higher probability, make the predictions much easier.

Assuming that we know that our data generating process is as described above, we can try to predict the next value based on a maximum likelihood estimation. We can compute the two likelihoods that the previous value was generated by the normal distribution with expected value $\mu_1 = 0$ and that it comes from the normal distribution with expected value $\mu_2 = 1$. We can then use as a simplified prediction, the mean value of the normal distribution with the higher likelihood.

Given a sequence of values $x_1, x_2, \dots, x_m$ and a sequence of states $\mu_{i_1}, \mu_{i_2}, \dots, \mu_{i_m}$ where $\mu_{i_j} \in \{0, 1\}$ in the automaton in figure 2, the likelihood for this sequence of states or paths is

$$\prod_{i=j}^m f(x_j | \mu_{i_j}) \cdot q_{j-1} \quad (12)$$

where $f(x|\mu)$ is the probability density function of the normal distribution with expected value μ and variance σ^2 and

$$q_{j-1} = \begin{cases} (1 - p) & \text{if } \mu_{i_j} = \mu_{i_{j-1}}, \\ p & \text{otherwise.} \end{cases} \quad (13)$$

q_0 is the probability for the initial state μ_{i_1} .

The likelihood for a sequence $x_1, x_2, \dots, x_m$ is then computed as the sum of the likelihoods over all paths.

$$P(\mu = c | x_1, \dots, x_m) = \sum_{paths} P(path) \quad (14)$$

The corresponding path trees for the likelihood computations are shown in figures 3 and 4 for the state corresponding to the normal distribution $N(0, \sigma^2)$ and the normal distribution $N(1, \sigma^2)$, respectively.

It is, of course impossible, to compute the exact likelihood as the sum over all possible paths for larger m , since there are 2^m possible paths. Therefore, we restrict the likelihood computation to the last k states, so that the likelihoods are actually computed “backwards” along the paths. In this way, we also do not require the probabilities for the initial states. The error for the differences in the two likelihoods will be quite small with this procedure, since the conditional likelihood for the two states k steps backwards given the final

parameters	Maximum likelihood estimation	TS Evolving Fuzzy Models
$\sigma = 0.5, p = 0.25$	0.6390	0.7116
$\sigma = 1.0, p = 0.1$	1.2792	2.1124
$\sigma = 1.5, p = 0.2$	2.6678	3.7769
$\sigma = 2.0, p = 0.1$	4.3743	6.1057
$\sigma = 3.0, p = 0.001$	9.1785	12.3010
$\sigma = 6.0, p = 0.001$	35.7901	50.3640

TABLE I
MSE

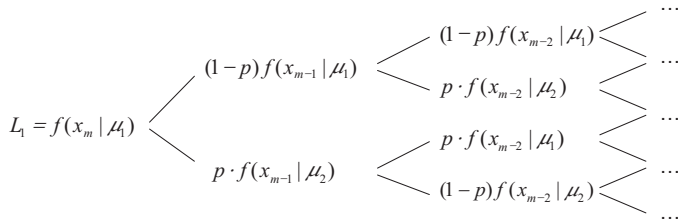


Fig. 3. Likelihood function for the state 1

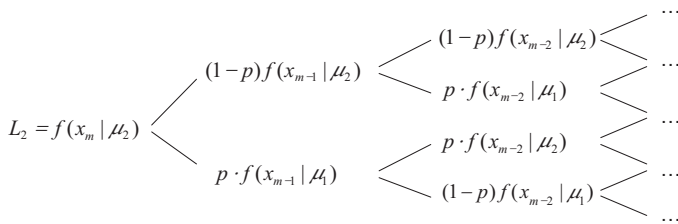


Fig. 4. Likelihood function for the state 2

state will be more or less independent of the final state for larger k according to the theory of Markov chains.

Table I shows a comparison of the described maximum likelihood estimation where we have chosen $k = 7$ with the Takagi Sugeno evolving fuzzy model described in [6]. In the case of the maximum likelihood estimation we simply predict the expected value of the normal distribution with the higher likelihood. This prediction could even be improved by choosing a weighted mean between the two expected values.

If we knew from which normal distribution the last value had been sampled, we could base our prediction on the knowledge that we sample from a mixture of two normal distributions. In this case, the expected quadratic error would be $\sigma^2 + p - p^2$. From table I we can see that for small values p , the mean square error of the maximum likelihood-based estimations is very close to theoretical lowest mean square error whereas the evolving system has a much larger error.

IV. CONCLUSIONS

We strongly argue in favour to also use benchmarks for evolving systems that are based on well-defined stochastic data generating processes. If only real world data are considered for benchmarks, it is not clear at all, how close an evolving system comes to the *unknown* best solution. With our simple stochastic models, we can explicitly provide or at least estimate the

best solution that can be achieved from a theoretical point of view, so that we have a clear measure, how close the evolving system can come to the best solution.

Of course, our comparisons are “unfair”, in the sense that we compare techniques from evolving systems, that do not make any specific assumptions about the data generating process, with estimators tailored to the specific problem. The assumption that we know the data generating process in principle and just need to estimate the parameters is unrealistic. Our intention is to give an impression of how much the evolving systems are biased to track the noise or randomness in the data generating process instead of learning the actual dependencies in the data.

Our examples were all restricted to the prediction of the next value. In terms of learning a function, the function was the identity function in the case of the random walks and binary function – providing only two possible outputs – in the case of the switching model. To make the situation more complicated, we could replace the identity function or the binary output by another function. We could also consider a number of our models in parallel to have multiple inputs and aggregate them by a simple function for the output.

We can also use more complicated models for the data generating process. But we should keep in mind that we need to have an idea, how well an optimized model could predict in order to have a comparison with the evolving system.

REFERENCES

- [1] Angelov, P., D. Filev, An Approach to On-line Identification of Evolving Takagi-Sugeno Models. IEEE Trans. Systems, Man, Cybernetics B, vol. 34, no. 1, 2004, pp. 484-498
- [2] Angelov, P., E. Lughofer, A Comparative Study of two Approaches for Data-Driven Design of Evolving Fuzzy Systems: eTS and FLEXFIS, Intern. Journal of General Systems, vol.37, issue 1, 2008, pp. 45-67.
- [3] Angelov, P., X. Zhou, F. Klawonn, Evolving Fuzzy Rule-based Classifiers, in: Proc. 2007 IEEE Symposium on Computational Intelligence in Image and Signal Processing (CIISP 2007). IEEE, Los Alamitos, 2007, pp. 220-225
- [4] Beringer, Hüllermeier. Efficient Instance-Based Learning on Data Streams. Intelligent Data Analysis, vol. 11, 2007, pp. 627-650
- [5] Klawonn, F., P. Angelov: Evolving Extended Naïve Bayes Classifiers. In: S. Tsumoto, C.W. Clifton, N. Zhong, X. Wu, J. Liu, B.W. Wah, Y.-M. Cheung: Proc. Sixth IEEE International Conference on Data Mining: Workshops. IEEE, Los Alamitos, 2006, pp. 643-647
- [6] Macias-Hernandez, J.J., P. Angelov, X. Zhou, Soft Sensor for Predicting Crude Oil Distillation Side Streams using Takagi Sugeno Evolving Fuzzy Models, Proc. 2007 IEEE International Conference on Systems, Man, and Cybernetics, Montreal, Canada, 2007, pp. 3305-3310.
- [7] Spitzer, F. Principles of Random Walk (2nd edition), Springer, Berlin, 2001

Clustering as a Tool for Self-generation of Intelligent Systems: A Survey

Rashmi Dutta Baruah, Plamen Angelov

Abstract – Fuzzy Rule Based (FRB) and Neuro-fuzzy systems are commonly used as a basis for intelligent systems due to their transparent and simple human interpretable structure. One of the crucial steps in designing FRB and neuro-fuzzy systems is to innovate the rule base. Data clustering is one of the approaches that have been applied extensively to automatically generate rules from input-output data. The goal of this paper is to critically review some of the most commonly used as well as recently developed clustering techniques, emphasizing their use in rule base generation. The paper explores the shift from offline clustering techniques to online and finally to evolving techniques that originated due to the current demand of adaptive systems.

I. INTRODUCTION

INTELLIGENT systems differ by other technical, social etc. systems by their ability to learn, reason, and make decisions.

It is common to represent technical and computer-based systems that have some degree of ‘*computational intelligence*’ by either a Fuzzy Rule Based (FRB) or neural-network system. FRB system, in particular has gained the attention of researchers and users due to their specific properties, one being their transparent and human interpretable rule-based structure that is expressive enough to represent even imprecise qualitative knowledge. The core components of such a system are: the rule base, and an associated inference process. The rules are of the form: IF *antecedent* THEN *consequent*, and the inference process determines the crisp output for a given input using the rule base.

FRB systems can be broadly classified into two families: Mamdani-type and Sugeno-type. In the Mamdani-type [1], also called linguistic systems, rules are represented as:

IF x_1 is A_1 and x_2 is A_2 and ... and x_n is A_n THEN y is B (1)

where x_i , $i = 1, 2, \dots, n$ is the input variable, A_i and B are linguistic terms (eg. *Small*, *Large*, *High*, *Low* etc.) defined by fuzzy sets, and y is the output associated with the given rule.

The rule structure of the second type, Sugeno-type [2], also called TSK type, is usually given as:

IF x_1 is A_1 and x_2 is A_2 and ... and x_n is A_n
THEN $y = a_0 + a_1 x_1 + \dots + a_n x_n$ (2)

where x_i , A_i , y are input variables, linguistic terms, and output variable associated with the rule respectively, and $a_0, a_1, \dots, a_n$ are consequence parameters.

Thus, in Mamdani-type the consequent of each rule is a fuzzy set whereas in Sugeno-type the consequent is a function of input variables. Due to this difference the inference mechanism of determining the output of the system in both the categories varies somewhat.

The early approaches to the design of the rule base involve representing the knowledge and experience of a human expert, associated with a particular system, in terms of IF-THEN rules. To achieve better system performance another alternative is to use expert knowledge as well as learn from system generated input-output data. This fusion of expert knowledge and data can be done in many ways. For example, one way is to combine linguistic rules from human expert and rules learnt from numerical data [3] and another way is to derive rules from expert knowledge and optimize the parameters (e.g. membership function) using input-output data by applying machine learning techniques [4, 5]. However, recently research in generating fuzzy rules only from input-output data has gained momentum in order to avoid the difficult task of knowledge acquisition [6]; moreover due to technological advancements huge amount of data is easily available.

System modelling requires structure identification and parameter identification. Structure identification deals with determining the input variables, number of rules etc. whereas parameter identification deals with antecedent parameters (membership functions of fuzzy sets) and consequence parameters. A clustering algorithm is mainly applied to structure identification (determining rules) by partitioning the data space. While most of the algorithms assume that input variables are available (based on data and prior knowledge or heuristics), others may optimize iteratively the input variables using various variable selection criteria [7-9]. There are numerous approaches for learning fuzzy rules from data such as, grid based [3, 10], neural network and neuro-fuzzy based [4, 11-13], and evolutionary computation based [14-16]. However, clustering techniques, especially fuzzy clustering, are being used extensively either independently or combined with other techniques for rule generation. Methods based on fuzzy clustering are appealing as there is a close connection between fuzzy clusters and fuzzy rules [17, 18].

This paper aims at reviewing various clustering techniques and highlighting the pros and cons of each technique in the context of fuzzy rule generation for FRB systems and similarly to neuro-fuzzy systems as examples of *computationally intelligent* systems. The techniques that are considered are

Both authors are with the Intelligent Systems Research Laboratory, Department of Communication Systems, InfoLab21, Lancaster University, Lancaster, LA1 4WA, UK; phone +44 (1524) 510525; fax: +44 (1524) 510493; e-mail: r.duttabaruah@lancaster.ac.uk

categorized here as: i) offline, ii) online, and iii) evolving. The remaining paper is organized as follows: section II presents an overview of fuzzy clustering and fuzzy rule generation, Section III, IV and V review the offline, online, and evolving clustering techniques respectively. Section VI presents a theoretical analysis of the algorithms discussed in the paper. Finally, section VII concludes the paper.

II. FUZZY CLUSTERING AND FUZZY RULE GENERATION

Given a data set, the aim of clustering is to partition it into different groups (clusters) so that the members in the same group are of similar nature, whereas members of different groups are dissimilar. While clustering, various similarity measures can be considered, one of the most commonly used is distance between data samples. A hard or crisp clustering technique, e.g. k-means [19], assigns a data sample to only one cluster whereas in fuzzy clustering a data sample can belong to all the clusters with certain degree of membership [20].

In order to generate fuzzy rules, fuzzy clustering can be done in input data space only, output data space only or jointly in input-output data space; each of these approaches has its own advantages and disadvantages. After clustering is applied, each cluster induces a rule by projecting the cluster to the respective coordinate space. For rules of Mamdani-type, Sugeno and Yasukawa [21] and Emami et.al. [22] used fuzzy clustering for clustering output data and then projecting the clusters on to the input space in order to define the rule premise. Babuška and Verbruggen [23] proposed a method to derive a linguistic model from a TSK model. The TSK model is determined by clustering in the input-output space and then the concept of complementary partition is applied to derive the linguistic model. The approach proposed by Salehfar et al. [24] is to apply clustering first to the output space then these clusters are projected on to the input variables. Again clustering is applied to these clusters in the input space to generate several sub-clusters to derive fuzzy rules.

In TSK type rules, the common approach is to apply clustering in the input-output data space and projecting the clusters on to the input variables coordinate to determine the premise (membership function) parameter of the rule [25-27] (Fig. 1). The consequent parameters of such rules may be estimated separately by using methods like least squares method.

The various clustering techniques used for learning fuzzy rules are categorized here into three categories; i) offline; ii) online, and iii) evolving clustering techniques depending on the mode of feeding data samples to them and the ability of the clustering structure to grow or shrink (to evolve) as opposed to the case when the number of clusters is pre-defined and fixed. Offline methods consider the entire data to be in the memory and perform multiple pass (iterations) over it to get the desired number of clusters (partitions). Such methods are simple and easy to implement compared to other techniques, however they are unable to handle high dimensional data. Further, when used for data partitioning and FRB (or neuro-fuzzy) *computationally intelligent* systems design, the resulting rule base is static and the system cannot handle any deviations in the input data which may be due to changes in the operating environment over time.

In order to incorporate such changes in the rule-base it is required to re-model the whole system [28]. As the technology has advanced, potentially a huge amount of data with a high data rate can be received from various applications such as packet monitoring in the IP network, real time surveillance systems, and sensor networks. Clustering such form of data, commonly referred as *data stream*, require the algorithms to be fast (non-iterative), memory efficient (need not store previously seen data), adaptive (change the model structure and parameters taking into account data *shift* and *drift*) [29, 30]. Online clustering techniques, as considered here, are algorithms that are incremental or one pass and can handle high dimensional data. However, many online clustering algorithms do still consider the structure of the clusters to be fixed (the number of clusters to be pre-defined) and only change the position of the cluster centres (e.g. Self-organizing maps SOM, Adaptive resonance theory ART [31], etc.). If, in addition to the fact that the data samples are provided in one pass, incrementally, and on-line, the cluster structure (number of clusters) can also change (grow or shrink) then we have *evolving* clustering [30]. Thus, evolving clustering, introduced in 2001-2002 [33-36] incorporates the features of online algorithms but in addition have the important property of evolving or adapting the model structure itself, which paves the way for complex systems autonomous structure identification which is a breakthrough in complex (including intelligent) systems design and learning. Later on various applications and developments were reported based on this concept.

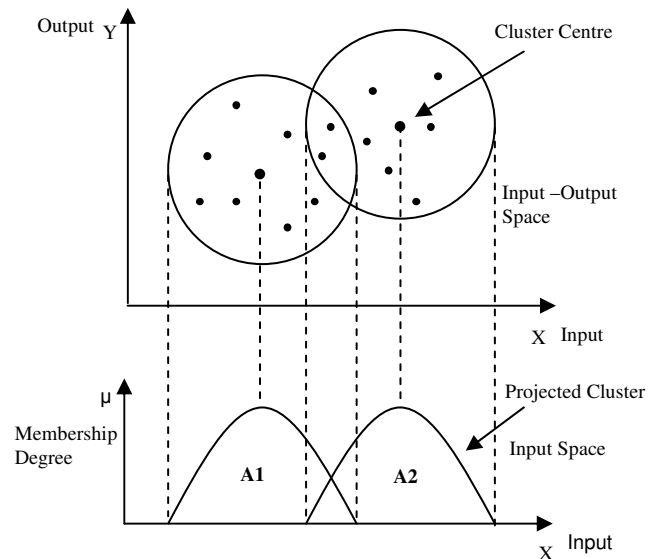


Fig. 1. Fuzzy clusters in input-output space and projection on to input space.

III. A BRIEF REVIEW OF THE OFF-LINE CLUSTERING METHODS IN RELATION TO FRB SYSTEMS DESIGN

Some of the most commonly used clustering techniques for offline FRB system design are: fuzzy c-means, Gustafson-Kessel algorithm, mountain clustering, and subtractive clustering.

Fuzzy c-means (FCM) [37, 38] is adapted from k-means algorithm and is based on minimization of an objective function (equation 3) to obtain optimal number of clusters. A $c \times n$ fuzzy

partition matrix U is defined, where n is the number of data points and c , $1 < c < n$ is the number of clusters. Each element of U represents the degree of membership (u_{ij}) of a j^{th} data point to i^{th} cluster. The goal is to minimize the squared distance of the data points to their cluster centers given the two conditions (equation 4a and 4b).

$$J(x, U, C) = \sum_{i=1}^c \sum_{j=1}^n u_{ij}^m \|x_j - c_i\|^2, \quad (3)$$

$$\sum_{j=1}^n u_{ij} > 0, \quad i = 1, \dots, c \quad (4a)$$

$$\sum_{i=1}^c u_{ij} = 1, \quad j = 1, \dots, n \quad (4b)$$

X is the data set and C is the set of cluster prototypes (usually cluster centers). The parameter m is called *fuzzifier* exponent, cluster boundaries become softer with higher values of m and vice versa. Before clustering process begins, c , m and U are initialized. A threshold value for terminating condition is also selected. After initialization, the cluster centers (equation 5) and then the partition matrix are updated (equation 6) iteratively. The process continues until the change in the partition matrix at step k and at step $k+1$ is less than a threshold value.

$$c_i = \frac{\sum_{j=1}^n u_{ij}^m x_j}{\sum_{j=1}^n u_{ij}^m} \quad (5)$$

$$u_{ij} = \frac{1}{\sum_{k=1}^c \left(\frac{\|x_j - c_i\|}{\|x_j - c_k\|} \right)^{\frac{2}{m-1}}} \quad (6)$$

The centre of a cluster is nothing but the mean of all points weighted by their degree of membership to the cluster. The algorithm attempts to move the cluster centers to the proper location within the data set by iteratively updating the cluster centers and the membership degrees. The fuzzy c-means algorithm is simple to implement and has been widely applied in isolation or in combination with other techniques in various domains [39-41]. However, the algorithm is sensitive to initialization of parameters and may get stuck in local minimum [20].

Many variants of fuzzy c-means algorithm have been proposed in the literature by applying various types of distance measure other than Euclidean distance. Gustafson-Kessel algorithm (GK) [32] uses Mahalanobis distance as a distance measure between a data point and a cluster centre in order to generate clusters of various size and shapes other than spherical clusters. Each cluster is characterized by cluster centre and covariance matrix (cluster prototype). The distance for i^{th} cluster is given by equation (7), where A_i is the covariance matrix of the cluster. This allows the adaptation of distance norm to the shape of each cluster. Similar to fuzzy c-means

algorithm, the cluster centers and the membership grades are updated iteratively in addition to the covariance matrix until change in the partition matrix at step k and at step $k+1$ is less than a threshold value.

$$D_{ij}^2 = (x_j - c_i)^T A_i^{-1} (x_j - c_i) \quad (7)$$

The eigenstructure of the cluster covariance matrix presents the shape and orientation information of the cluster. If the matrix is restricted to diagonal matrix then axis-parallel clusters are generated that are suitable for fuzzy-rule generation [42]. The disadvantage of this algorithm is that it is computationally more intensive compared to fuzzy c-means due to involvement of matrix inverse calculations while updating the covariance matrix. Moreover, it is sensitive to initialization of parameters [20].

Mountain clustering [43, 44] is a simple algorithm that can be used either with fuzzy c-means to generate initial cluster centers or independently to generate approximate cluster centers. Each dimension of the data space is discretized into equidistant points forming a grid. The intersection of the grid lines are called *nodes* and are the potential clusters. A *mountain function* is defined that is related to the density of neighbouring data points and is used to calculate the potential of each grid point (node) to become a cluster centre. The value of the function is high for a node with many neighbouring data points. For all the nodes the mountain function is calculated and the node with highest value is selected as the first cluster centre. To determine the next cluster centre, an amount proportional to the distance of the point to the first cluster centre is subtracted from the current mountain function value of each of the nodes. Thus, the nodes near to the first cluster centre will have higher reduction of their value as compared to the distant nodes. This ensures that nodes closer to the cluster centre are not selected as new cluster centers. Now, the node with the highest remaining mountain function value is chosen as the next cluster centre. This process of selecting cluster centers and subsequently reducing the mountain function value continues until the value of current maximum of the mountain function compared to the first maximum falls below a threshold. The algorithm is simple, however computationally expensive for high dimensional data. Each iteration requires evaluation of $O(n^d)$ nodes (considering equal number of grid lines in all dimensions) where n is the number of grid lines and d is dimension of data space. Further, the generation of number of clusters is sensitive to grid resolution, the finer the grid lines more are the potential cluster centers (nodes) i.e. a tradeoff between accuracy and computational complexity. Also, the method needs to predefine certain critical parameters for calculation of mountain function and a threshold value as a terminating criterion.

Subtractive clustering [27, 45] is an improved version of mountain method for cluster estimation. The important difference between the two methods is that, data points are considered as potential clusters instead of grid points in subtractive clustering. This method also assumes that the data points are normalized and bounded by a hypercube. For every data point a *potential* value is calculated and the point with the highest potential value is selected as the first cluster centre. The potential value is dependent on the distance of the data point to

all other data points, i.e. the larger the number of neighbouring data points the higher is the potential. The neighbourhood of a data point is defined by a constant (radius r); data points outside the neighbourhood do not have significant influence on the potential value. Similar to the mountain method, the next step is to reduce the potential of all data points by an amount that is dependent on their distance to the cluster centre. So, the points closer to the cluster centre have less chance to be selected as next cluster centre. Now, the next cluster centre is the point with the remaining maximum potential. Two threshold values are defined that controls the termination of the clustering process. If the ratio of potential (P_k) of the current data point (x) and the potential of the first cluster centre (P_1) is greater than an *upper threshold* value ($P_k / P_1 > uth$) then x is accepted as cluster centre and the process continues. If this ratio is less than a *lower threshold* value ($P_k / P_1 < lth$) then x is rejected and the process terminates. If the ratio lies between the two threshold values then the smallest distance (*minDist*) between x and existing clusters is determined and the following condition is examined: if sum of the *minDist*/ r and P_k / P_1 is greater than or equal to , ($minDist/r + P_k/P_1 \geq 1$) then x is set as new cluster centre and the process continues else it is rejected and data point with the next highest potential is selected and tested for above conditions. Although, the computational complexity increases linearly with the dimension of the data set, it grows as the square of the number of samples. In most cases it has not been tested on large data sets [46, 47]. Further, this algorithm also needs to predefine certain critical parameters required for potential calculation, neighbourhood definition and threshold values.

Fuzzy c-means and Gustafson-Kessel algorithms are distance-based whereas Mountain clustering and Subtractive clustering algorithms are density based and therefore, the later algorithms can be modified suitably for online fuzzy rule generation as discussed in section v.

IV. A REVIEW ON ONLINE CLUSTERING METHODS SUITABLE FOR DESIGN OF FRB SYSTEMS

The area of online clustering or data stream clustering itself is an emerging and wide area of research engaging many researchers. The literature provides numerous methods that have been proposed for clustering data streams [48, 49]. In this section a brief review of selected online clustering methods is presented.

Several variations of fuzzy c-means algorithm have been proposed for data streams [50-53]. Both the algorithms streaming fuzzy c means (SFCM) [50] and online fuzzy c means (OFCM) [51] assumes that the data is arriving and processed in batch i.e. n_1 data points arrive at time T_1 , n_2 at time T_2 and so on. At the initial step the number of clusters is predefined. The first batch of data is clustered using FCM and only the weighted cluster centers are stored. The weight of a cluster center is the sum of membership degrees of all data to that cluster. As the next batch of data arrives the cluster centers of previous batch of data are clustered together with this new set of data. The weights are now calculated using the membership values of current data points. Thus, the clustering of each batch of data points is initialized with the centers obtained from the previous clustering (*cluster history*). The equation for calculation of

cluster centre, membership function, and the objective function of FCM are modified to incorporate the weight factor. The OFCM is better compared to SFCM in that the result of SFCM is dependent on the clustering history. The aim of sWFCM (Weighted Fuzzy C-Means for data stream) algorithm [52] is to reduce the memory usage as compared to FCM. The basic approach of sWFCM algorithm is similar to the algorithms described above. However, it uses a different measure of weight. Each data is associated with a time weight factor that represents the data's influence extent on the clustering process. The algorithm iteratively updates the weighted cluster centers till objective function value reaches a required minimum or the number of iterations reaches a threshold value. Although these algorithms take into account the efficient memory usage, they are more suitable for large data sets rather than high speed real time online data due to involvement of iterative computations.

The adjustable fuzzy c-means algorithm [53] considers that the incoming data is available as snapshots (chunks of data) and intends to adjust the number of clusters dynamically for each snapshot. The cluster prototypes generated in one data snapshot are successively migrated to the next data snapshot (chunk). These prototypes from previous snapshot are used as starting point for clustering process in the current data snapshot. The adjustment of number of clusters is performed by cluster splitting or merging based on a predefined threshold value. The property of dynamically adjusting the number of clusters depending on the underlying data makes this approach more appealing compared to other modified FCM methods.

A variant of Gustafson-Kessel algorithm for evolving data stream is proposed in [54]. It assumes an initial set of clusters that are obtained by applying GK algorithm offline. For each incoming data point, its distance to all the existing clusters is calculated. If the distance is less than or equal to the radius of the nearest cluster then the data point is assigned to the cluster, here the radius is the distance between the cluster centre and farthest point belonging to the cluster such that its membership degree is greater than or equal to a given membership degree threshold. In this case the cluster centre is updated using Kohonen rule, and the inverse covariance matrix and its determinant are updated using Woodbury matrix inversion lemma and a learning approach [54, 55]. If the distance of the data point is greater than the nearest cluster radius then a new cluster is created considering the data point as its centre. Its covariance matrix is initialized to the covariance matrix of the closest cluster. New clusters are accepted depending on a threshold value. The lower bound of the threshold value depends on the dimension of the data and is estimated from the minimum number of points required to learn the covariance matrix parameters. High value of this threshold parameter, in turn, discards the outliers. Although, some offline processing is required to initiate the algorithm, it does not require data snapshots/chunks later. Further, it is not iterative and automatically detects outliers.

Mean shift algorithm [56], a kernel density estimation based nonparametric technique, is capable of determining clusters with no restrictions on their shape. It uses the *mean shift* procedure [57, 58] to find the point of maximum density. The data points in the d -dimensional feature space are handled

through their empirical probability density function (pdf) where dense region in the feature space corresponds to local maxima or mode of the distribution. This approach provides the number of clusters (modes of the pdf) automatically, but is iterative. However, if appropriately modified for online clustering, it may be used for FRB system design.

V. EVOLVING CLUSTERING METHODS - A BASIS FOR ON-LINE AUTONOMOUS SELF-DEVELOPMENT OF FRB SYSTEMS

As defined in [30,59] the term *evolving* is not similar to the term *evolutionary* as the former is related to the life-long self-development of an individual entity while the later is concerned towards generation of population of individuals by reproduction, mutation, and natural selection etc. The development of evolving techniques that are adaptive was motivated by the need to design dynamic systems that can continuously change over time by learning from interactions with the environment, and self-monitoring.

An online clustering technique for adapting TSK fuzzy rule base is presented in [33, 34]. The method builds upon subtractive clustering and uses proximity-based *potential* value to determine a cluster centre. A Cauchy function (given in equation (8)), which is monotonic and inversely proportional to the distance between all of the data points is used to determine the potential.

$$P_i = \frac{1}{1 + \frac{1}{N} \sum_{j=1}^N \sum_{k=1}^{n+1} (d_{ij}^k)^2}, \quad i = 1, 2, \dots, N \quad (8)$$

where $d_{ij}^k = z_i^k - z_j^k$, denotes the projection of the distance between the two data points z_i^k and z_j^k on the axis z^k ; $k = 1, 2, \dots, n+1$; $z \in \mathbb{R}^{n+1}$ and N is the number of training data samples.

For a given data point, the higher the number of surrounding neighbours the higher its potential value is. At first step, the first data sample of the input stream is established as the first cluster centre with potential set to one. As the next sample arrives its potential is calculated using a recursive form of equation (8). Since potential depends on the distance to all the data points, arrival of a new sample causes the potential of all the cluster centres to change. The potential of the new data point is compared with the potentials of all the existing cluster centres and one of the following actions is performed: (i) the new data is added as a new cluster centre if it has the highest potential compared to all the existing cluster centres; (ii) if the new data point has the highest potential and it is near to a cluster centre then it replaces the later. If both conditions are not satisfied then the data is added to the cluster with closest cluster centre and then next data sample in the stream is considered. The process continues till all the samples in the data stream have been considered. Each of the cluster centres represents a rule antecedent. Thus, with each incoming data, as the cluster centre is updated the rule antecedent automatically gets updated. One of the favourable characteristics of this algorithm is that it automatically handles the outlying data because the potential of such data would be low due to their distance from the normal data. Further, it does not require any user-defined threshold values or parameters like number of clusters etc. that are

required in other clustering techniques like subtractive or mountain clustering. However, Cauchy type potential recursive calculation is crucial. The proposed algorithm has been applied for identification of evolving FRB models [60] by incorporating a threshold value. The decision of introducing a new cluster centre (rule) or replacing an existing cluster centre is considered if the potential of a new data sample is higher than the threshold value. The consequent parameters of the rules are determined using recursive least square (RLS) technique.

An evolving neural network (eNN) model, linked to the evolving rule base (eR) model, is introduced in [35]. The structure of this neural network evolves (new neurons are added or old ones are replaced) as the new data arrives and depending on the data sample's *potential*. The proposed eNN has six layers where layer 1 is input layer, layer 2 corresponds to fuzzy set in a TSK rule, layer 3 represents set of rules where each neuron represents a fuzzy rule, layer 4 represents the consequent part of a TSK rule, layer 5 aggregates the antecedent and consequent of each rule, and layer 6 generates the final output of the system. Online clustering is used to determine the cluster centres that represent a neuron in layer 3 of eNN (focal point of a rule). The condition that decides addition of a new neuron or replacement of an old neuron is similar to the one described above [33] except that the potential of a new data point is calculated recursively using a recursive form of equation (9).

$$P_k = e^{-\alpha \sum_{i=1}^k \sum_{j=1}^{n+1} (z_{kj} - z_{ij})^2} \quad \text{and} \quad \alpha = \frac{4}{r_a^2} \quad (9)$$

where $z^T = [x^T; y]$ denotes augmented data vector and r_a is the radius defining the neighbourhood in a cluster [27].

A simplified approach for learning evolving TS fuzzy models is (Simple_eTS) presented in [61]. The basic approach of determining cluster centres is similar to [33, 34] (as described in paragraph 2, section V) with a simplified measure called *scatter* instead of *potential*. The recursive calculation of scatter is more efficient compared to potential. A method based on *population* (number of data samples assigned to a cluster) is also proposed that can be used to reduce the rule base. A rule/cluster is ignored by setting its firing level to 0 if the value of population is less than 1% of the total data samples. Thus, the rule base also evolves incrementally as the data arrives.

In all the methods described above the radius of the cluster is not adaptive, rather it is predefined. The eClustering (evolving clustering) technique proposed in [62] alleviates this drawback. It is applied to the generation of evolving extended TSK type system (exTS) from data streams. The extended TSK systems are multi-input multi-output (MIMO) and a combination of both zero and first order TSK type systems. The clustering is applied to partition the input-output joint data space to retrieve the antecedent part of the fuzzy rules. A Gaussian membership function defined in equation (10) is used.

$$u_j^i = e^{-\frac{\|x_j - x^{i*}\|^2}{2(r_j^i)^2}} \quad (10)$$

where u_j^i is the membership degree of the j^{th} input x_j , $j = 1, \dots, N$ to the i^{th} fuzzy set $i = 1, \dots, n$; x^{i*} is the focal point of the i^{th} rule antecedent; r_j^i is the spread of the membership function.

From equation (10) it can be observed that two parameters are required to be determined by clustering, the cluster centre (x^{i*}) (focal point of a fuzzy rule) and the radius of cluster (spread of the membership function). The algorithm is an extension of the on-line version of the subtractive clustering used in [33, 34] and the basic steps are similar to the methods described above with the following developments. Most importantly, it adapts the cluster radius based on the local spatial density and this influences the fuzzy sets of the antecedent part of a rule when the clusters are projected on to the input variables axes. Two measures, *support* and *age*, are described that can be used along with the value of the *potential* to replace a cluster centre (respectively, rule). Support of a cluster/rule is the number of data points within the radius of a cluster centre (same as *population*). The rules with very low support can be ignored. Age is the difference between the number of data samples and the average sum of the time indices of the data sample for a given cluster. Thus, the value of age of a cluster is in the range $(0; k]$ and it determines whether a cluster is *young* (values close to 0) or *old* (value close to k). If a cluster is *young* it means recent data is included in the cluster. So, new data with high *potential* value can replace *old* clusters.

Another algorithm that can be applied to dynamic clustering of stream of data where the number of clusters is not required to be defined *a priori* is the Evolving Clustering Method (ECM) [36]. At the initial step of ECM, the first input data sample is considered as first cluster with the data itself as the cluster centre and the cluster radius set to zero. A threshold value (Dth) is also defined that limits the cluster size i.e. the cluster radius can grow only up to this threshold value. As the next sample arrives, the Euclidean distance ($dist$) between this sample and all other existing cluster centres is determined. Based on the following three conditions either the data sample is included in an existing cluster with or without any update, or a new cluster is created: (i) the cluster with minimum distance ($mindist$) to the sample is selected, if $mindist$ is less than the radius of this cluster then the sample is included in the cluster and no updates are required; (ii) for every existing cluster the respective $dist$ is added to the radius (let this value be $range$), the cluster with minimum $range$ is selected, if $range$ is less than twice Dth then the sample belongs to this cluster, the radius of this cluster is updated to $range/2$ and the cluster centre is updated by positioning it in the line joining data sample and cluster centre so that now the distance between the new centre and the sample is equal to the new radius value; (iii) if $range$ is greater than twice Dth then a new cluster is created with the input data sample as the cluster centre. This process continues till all the data samples in the input stream are processed. ECM has been applied to Dynamic Evolving Neural-Fuzzy Inference System (DENFIS) [36], a TSK neuro-fuzzy inference system that evolves by taking into account the variations in the input data. In DENFIS, as in a typical evolving system fuzzy rules are generated and updated while the system operates. The ECM method is applied to the input data space to determine the fuzzy sets in the antecedent part of the rules. The consequence of the rules is determined by applying RLS estimator. Several variants of ECM are also present like fuzzy ECM [63] and ECM for classification [64]. Although the ECM is a simple method,

outlier detection is not an integral part of the clustering process. Also, it requires threshold to be specified and usually generates a large number of clusters (because it is distance-based, not density based) that later needs to be pruned.

In FLEXFIS (Flexible Fuzzy Inference System) [65] an incremental clustering approach is used for partitioning the input-output data space. The data are assumed to be normalized to the unit interval in each dimension of data space forming a hypercube. The clustering technique is a modified version of vector quantization (VQ) that incorporates *vigilance parameter* (from Adaptive Resonance Theory, ART [66]) for update of cluster centres. The vigilance parameter is chosen to be proportional to the diagonal of the n -dimensional data space. The number of clusters and the zone of influence (spread of the fuzzy sets) are not required to be predefined and are generated incrementally. One of the distinct features of this algorithm is that it selects the nearest cluster to a new data point by comparing the distance of this new data point to the surface (instead of centre) of all the existing clusters. This feature and the vigilance parameter together avoid *over-clustering*. Initially the number of clusters is set to 0. As the first data sample arrives, it is set as the new cluster centre. For subsequent data samples one of the following steps is performed: (i) if the current data sample (x) is inside the range of influence of any cluster then its distance to the cluster centre of all such clusters is calculated. The cluster centre with minimum distance to x is selected and all its components along with the variance are updated by moving it towards the data sample (x). (ii) if x lies outside the range of influence of all the clusters then its distance to the cluster centre of all the clusters is calculated. The cluster centre with the minimum distance to x is selected and if this distance is greater or equal to the vigilance parameter, a new cluster is created with centre as x . The process continues till data is available. Although the clustering method used in FLEXFIS do not require certain parameters to be predefined (initial value of cluster numbers, cluster radius), it needs other parameters for old cluster centre shifting and new cluster generation. Moreover, the outliers are not directly addressed by this clustering algorithm.

VI. THEORETICAL ANALYSIS OF THE ALGORITHMS

The offline, online, and evolving algorithms are listed in Table I in decreasing order of computational cost and memory requirements. The computation time for FCM is $O(c^2nd)$ per iteration, where c is the number of clusters, n is the number of data samples, and d is the data dimension. It requires space for storing the entire data set as well as the partition matrix i.e. $O(nd+nc)$. The computational cost for G-K algorithm is $O(cnd(d+c))$ per iteration considering $d < c < n$, and in addition to entire data set and the partition matrix, it requires to store covariance matrix for each cluster centre. In mountain clustering method, the computation time depends on the number of grid nodes. The number of grid nodes in turn depends on the resolution of the grid and the data dimension. It needs to store the grid nodes along with the data set. In subtractive algorithm the computing time for calculation of potential values is $O(n^2d)$ and subsequently the reduction of potential takes $n \times d$ time per iteration. The space requirement is comparatively low as it

needs to store only the data set. The online modified versions of FCM algorithm (SFCM, OFCM, sWFCM) require $O(c^2 n_i d)$ time per iteration per data chunk, considering the number of weighted centers $\ll n_i$ (the size of the i^{th} data chunk). While processing, they need to store the data chunk, the weighted centres, and the partition matrix. The online version of the G-K algorithm requires the same time as G-K algorithm in the initialization phase, and subsequently most of the time is required for updating the partition matrix for a data sample. The space requirement is to store the initialization data, and for subsequent data sample it needs to store the current sample, the covariance matrices and the partition matrix. In adjustable FCM, the time requirement for a data chunk is slightly more than FCM if cluster splitting and merging takes place. The evolving algorithms (eClustering, ECM, clustering in FLEXFIS) require $O(cd)$ time per sample, and needs to store the current sample.

The theoretical analysis of the computation time and space requirement of offline algorithms indicate that, assuming the same number of iterations, the computational cost of FCM is least and subtractive clustering has the lowest memory requirements. The online methods that are discussed here have almost the same computational time and memory requirements. Also, all the evolving methods incur the same computational time and memory requirement. Further, it is self-explanatory that the offline algorithms are computationally the most intensive with high memory requirements, and the evolving algorithms are the least.

VII. CONCLUSIONS

One of the phases of intelligent system design based on fuzzy rule base or neuro-fuzzy model is rule generation. At present the most preferred way is to generate the rules automatically from the input-output data using data clustering. The paper has provided an overview of around fifteen clustering algorithms in the context of fuzzy rule generation and under the category of offline, online, and evolving clustering techniques. Most of the offline techniques are iterative and applies the clustering process over all the training data. Therefore, such techniques are not suitable for design of intelligent systems that continuously accept data from various sources and need to improve in terms of performance over a period of time. Using online techniques in such scenarios is intuitive as they are efficient compared to offline techniques in terms of computation and memory usage. However, there remains a huge scope for improvement of such techniques to conform them to adaptive intelligent system design. An attempt to meet such requirements is evolving techniques that have been applied to design adaptive FRB or Neuro-fuzzy systems. Such clustering methods can still be improved, such as by reducing the number of user-defined parameters, incorporating mechanisms to detect outlying data so that irrelevant clusters are not formed. Further, in most cases such techniques have been applied in scenarios where output data corresponding to an input data is available. The design of adaptive intelligent systems using evolving techniques in fully unsupervised scenarios is still an open issue.

TABLE I
ALGORITHMS IN DECREASING ORDER OF COMPUTATIONAL COST AND MEMORY REQUIREMENT

Algorithm	Computational costs	Memory requirements
Offline	Mountain clustering → Subtractive clustering → G-K → FCM	Mountain clustering → G-K → FCM → Subtractive clustering
Online	online G-K → adjustable FCM → SFCM ≈ OFCM ≈ sWFCM	online G-K → SFCM ≈ OFCM ≈ sWFCM ≈ adjustable FCM
Evolving	FLEXFIS clustering ≈ eClustering ≈ ECM	eClustering ≈ FLEXFIS clustering ≈ ECM

REFERENCES

- [1] E.H. Mamdani, "Application of fuzzy logic to approximate reasoning using linguistic systems", *Fuzzy Sets and Systems*, vol. 26, pp. 1182-1191, 1997.
- [2] T. Takagi and M. Sugeno, "Fuzzy identification of systems and its application to modelling and control," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, No. 1, pp. 116-132, 1985.
- [3] Li-Xin Wang and J.M. Mendel, "Generating fuzzy rules by learning from examples," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 22, No.6, pp. 1414-1427, 1992.
- [4] J.-S.R. Jang, "ANFIS: Adaptive-network-based fuzzy inference systems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, No. 3, pp. 665-685, 1993.
- [5] J.-S.R. Jang and C.-T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Transactions on Neural Networks*, vol. 4, No. 1, pp. 156-159, 1993.
- [6] K.L. McGraw and K.H. Harbison-Briggs, "Knowledge Acquisition: Principles and Guidelines," New Jersey: Prentice Hall, Englewood Cliffs, 1989.
- [7] T.-P. Hong and J.-B. Chen, "Finding relevant attributes and membership functions," *Fuzzy Sets and systems*, vol. 103, pp. 389-404, 1999.
- [8] N.R. Pal, "Soft Computing for feature analysis," *Fuzzy Sets and Systems*, vol. 103, pp. 201-221, 1999.
- [9] Y. Lin and G.A. Cunningham, "A fuzzy approach to input variable identification," in *Proceedings of IEEE Conference on Fuzzy Systems* 1994, pp. 2031-2036.
- [10] H. Ishibuchi, K. Nozaki, H. Tanaka, Y. Hosaka, and M. Matsuda, "Empirical study on learning in fuzzy systems by rice test analysis," *Fuzzy Set and Systems*, vol. 64, pp. 129-144, 1994.
- [11] S. Mitra and Y. Hayashi, "Neuro-Fuzzy generation: Survey in Soft Computing Framework," *IEEE Transactions on Neural Networks*, vol. 11, No. 3, pp. 748-768, 2000.
- [12] J. Moody and C. Darken, "Fast learning in networks of locally-tuned process units," *Neural Computing*, vol. 1, pp. 281-294, 1989.
- [13] K.B. Cho and B.H. Wang, "Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction," *Fuzzy Sets and Systems*, vol. 83, pp. 325-339, 1996.
- [14] P. Angelov, R. Buswell, "Automatic generation of fuzzy rule-based models from data by genetic algorithms," *Information Sciences—Informatics and Computer Science*, vol. 150, Issue1-2, pp. 17-31, 2003.
- [15] O. Cordón, F. Gomide et al., "Ten years of genetic fuzzy systems: Current framework and new trends," *Fuzzy Sets and Systems*, vol. 141, No. 1, pp. 5-31, 2004.
- [16] C.-K. Chiang, H.-Y. Chung, and J.-J. Lin, "A self-learning fuzzy logic controller using genetic algorithms with reinforcements," *IEEE Transactions on Fuzzy Systems*, vol. 5, No. 3, pp. 460-467, 1997.
- [17] F. Klawonn, "Fuzzy Sets and Vague Environments," *Fuzzy Sets and systems*, vol. 66, pp 207-221, 1994.
- [18] R. Kruse, J. Gebhardt, F. Klawonn, "Foundations of Fuzzy Systems," Wiley, Chichester, 1994.
- [19] T. Hastie, R. Tibshirani, J. Friedman, "The Elements of Statistical Learning Data mining, Inference, and Prediction," Second Edition, Springer, 2009, ISBN: 978-0-387-84857-0.
- [20] J.V. de Oliveira and W. Pedrycz (eds.), "Advances in Fuzzy Clustering and its Applications," Wiley, 2007, ISBN: 978-0-470-02760-8 (HB).

- [21] M. Sugeno and T. Yasukawa, "A fuzzy-logic based approach to qualitative modelling," *IEEE Transactions on Fuzzy Systems*, vol. 1, pp. 7-31, 1993.
- [22] M.R. Emami, I.B. Türksen, and A.A. Goldenberg, "Development of a systematic methodology of fuzzy logic modeling," *IEEE Transactions on Fuzzy Systems*, vol. 6, pp. 346-361, 1998.
- [23] R. Babuška and H.B. Verbruggen, "A new identification method for linguistic fuzzy models," in *Proceedings of IEEE International Joint Conference on Fuzzy Systems and Fuzzy Engineering Symposium*, Yokohama, Japan, 20-24 March, 1995, vol. 2, pp. 905-912.
- [24] H. Salehfar, N. Bengiamin, and J. Huang, "A systematic approach to linguistic fuzzy modeling based on input-output data," in *Proceedings of the 2000 Winter Simulation Conference*, Orlando, FL, USA, 10-13 Dec, 2000, vol. 1, pp. 480-486.
- [25] R. Babuška and H.B. Verbruggen, "An overview of fuzzy modelling for control," *Control Engineering Practice*, vol. 4, No. 11, 1593-1606-1996.
- [26] J. Zhao, V. Wertz and R. Gorez, "A fuzzy clustering method for the identification of fuzzy models for dynamical systems," in *Proceedings of 9th IEEE International Symposium on Intelligent Control*, Columbus, Ohio, USA, 1994.
- [27] S.L. Chiu, "Fuzzy model identification based on cluster estimation," *Journal of Intelligent Fuzzy Systems*, vol. 2, pp. 267-278, 1994.
- [28] P. Angelov, "Evolving rule-based models- A tool for design of flexible adaptive systems," Physica-Verlag, 2002, ISBN: 3-7908-1457-1.
- [29] P. Angelov and A. Kordon, "Adaptive inferential sensors based on evolving fuzzy models," *IEEE Transactions on Systems, Man, and Cybernetics -B*, pp. 1-11, 2009.
- [30] P. Angelov, X. Zhou, "Evolving fuzzy-rule-based classifiers from data streams," *IEEE Transactions on Fuzzy Systems*, vol. 16, No. 6, pp. 1462-1475, 2008.
- [31] G.A. Carpenter and S. Grossberg, "Adaptive Resonance Theory, Adaptive resonance theory (ART), in *The Handbook of Brain Theory and Neural Networks*," M.A. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 87-90.
- [32] D. Gustafsson and W. Kessel, "Fuzzy Clustering with a Fuzzy Covariance Matrix", in *Proceedings of the IEEE CDC, San Diego, California*, IEEE Press, Piscataway, New Jersey, 1979, pp.761-766.
- [33] P. Angelov, R. Buswell, "Evolving Rule-based Models: A Tool for Intelligent Adaptation," *9th IFSA World Congress*, Vancouver, BC, Canada, 25-28 July 2001, pp.1062-1067.
- [34] P. Angelov, An Approach for Rule-base Adaptation using On-line Clustering, *2nd EUNITE Conference*, 19-22 September 2002, Albufeira, Portugal, pp. 47-52.
- [35] P. Angelov, D. Filev, "Flexible Models with Evolving Structure," *IEEE Symposium on Intelligent Systems*, Varna, Bulgaria, 10-12 September 2002, v.2, pp.28-3.
- [36] N. Kasabov and Q. Song, "DENFIS: dynamic evolving neural-fuzzy inference system and its application for time-series prediction," *IEEE Transactions on Fuzzy Systems*, 2002, vol. 10, pp. 144-154.
- [37] J.C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact, well-separated clusters," *Journal of Cybernetics*, vol. 3, pp. 32-57, 1974.
- [38] J. Bezdek, "Pattern Recognition with fuzzy objective function algorithms," Plenum Press, New York, 1981.
- [39] H.S. Hwang, K.B. Woo, "Linguistic fuzzy model identification", *IEEE proceedings-control theory and applications*, vol. 142, Issue 6, pp. 537-544, No. 1995.
- [40] Y. Yingchun, Z. Laibin, L. Wei, W. Zhaohui, "Fuzzy C-means algorithm in work condition recognition of oil pipeline", *4th IEEE conference on Industrial Electronics and Applications*, 25-27 May 2009, pp. 682-686.
- [41] Hyong-Euk Lee, Kwang-Hyun Park, Z.Z. Bien, "Iterative fuzzy clustering algorithm with supervision to construct probabilistic fuzzy rule base from numerical data", *IEEE Transactions on Fuzzy Systems*, vol. 16, Issue 1, pp. 263-277, Feb. 2008.
- [42] F. Höppner, F. Klawonn, R. Kruse and T. Runkler, "Fuzzy Cluster Analysis", J. Wiley & Sons, Chichester, United Kingdom, 1999.
- [43] R. R. Yager and D. P. Filev, "Approximate cluster via the mountain method", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, No. 8, pp. 1279-1284, 1994.
- [44] R.R. Yager and D.P. Filev, "Learning of fuzzy rules by mountain clustering", in *Proceedings of SPIE Conference on Applications of Fuzzy Logic Technology*, Boston, MA, pp. 246-254, 1993.
- [45] S.L. Chiu, "Extracting fuzzy rules from data for function approximation and pattern classification", Chapter 9 in *Fuzzy Information Engineering: A guided Tour of Applications*, D. Dubois, H. Prade, and R. Yager (Ed.), John Wiley & Sons, 1997.
- [46] S. Chopra, R. Mitra, V. Kumar, "Identification of rules using subtractive clustering with application to fuzzy controllers", in *Proceedings of the 3rd International Conference on Machine Learning and Cybernetics*, Shanghai, 26-29 Aug 2004, pp. 4125-4130, 2004.
- [47] K. Demirti and M. Khoshnejad, "Autonomous parallel parking of a car-like mobile robot by a neuro-fuzzy sensor-based controller", *Fuzzy Sets and Systems*, vol. 160, pp. 2876-2891, 2009.
- [48] M.M. Gaber, A. Zaslavsky, S. Krishnaswamy, "Mining data streams: A Review", *ACM SIGMOD Record*, vol. 34, Issue, pp. 18-26, 2005.
- [49] Charu C. Aggarwal (ed.), *Data Streams: Models and algorithms*, Springer, 2007.
- [50] P. Hore, L.O. Hall and D.B. Goldgof, "A fuzzy c means variant for clustering evolving data streams", *IEEE International Conference on Systems, Man and Cybernetics, Montreal*, October 2007, pp. 360-365.
- [51] P. Hore, L.O. Hall, D.B. Goldgof, W. Cheng, "Online fuzzy c means", *Annual Meeting of the North American Fuzzy Information Processing Society*, 19-22 May 2008, pp. 1-5.
- [52] R. Wan, X. Yan and X. Su, "A weighted fuzzy clustering algorithm for data stream", *2008 ISECS International Colloquium on Computing, Communication, Control and Management*, 3-4 August 2008, vol. 1, pp. 360 – 364, 2008.
- [53] W. Pedrycz, "A dynamic data granulation through adjustable fuzzy clustering", *Pattern Recognition Letters*, Elsevier, vol. 29, pp. 2059-2066, 2008.
- [54] O. Georgieva and D. Filev, "Gustafson-Kessel algorithm for evolving data stream clustering", in *Proceedings of International Conference on Computer Systems and Technologies 2009*, pp. IIIB.14-1-6.
- [55] T. Kohonen, "Self Organization and associative memory", 3rd Edition, Springer Information Sciences Series, 1989, ISBN: 0-387-51387-6.
- [56] D. Comaniciu and P. Meer, "Mean Shift: A robust approach toward feature space analysis", *IEEE Transactions on Pattern Analysis and machine Intelligence*, vol. 24, No. 5, pp. 603-619, 2002.
- [57] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition", *IEEE Transactions on Information Theory*, vol. 21(1), pp. 32-40, 1975.
- [58] Y. Cheng, "Mean shift, mode seeking, and clustering", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17(8), pp.790-799, 1995.
- [59] N. Kasabov, "Evolving Connectionist Systems Methods and Applications in Bioinformatics, Brain Study and Intelligent Machines", Springer-Verlag London, 2003, ISBN: 85233-400-2.
- [60] P. Angelov and R. Buswell, Identification of Evolving Fuzzy Rule-Based Models", *IEEE Transactions on Fuzzy Systems*, vol. 10, No. 5, pp. 667-677, 2002.
- [61] P. Angelov and D. Filev, "Simple_eTS: A simplified method for learning evolving Takagi-Sugeno fuzzy models", in *Proceedings of 2005 IEEE International Conference on Fuzzy Systems*, Reno, NE, USA, 2005, pp. 1068-1073.
- [62] P. Angelov and X. Zhou, "Evolving fuzzy systems from data streams in real-time", *2006 International Symposium on Evolving Fuzzy Systems*, September, 2006, pp. 29-35.
- [63] V. Ravi, E.R. Srinivas and N.K. Kasabov, "On-line evolving fuzzy clustering", in *Proceedings of International conference in Computer Intelligence Multimedia Applications*, 13-15 Dec. 2007, vol. 1, pp. 347-351.
- [64] Q. Song and N. Kasabov, "Weighted data normalization and feature selection for evolving connectionist systems", in *Proceedings of 8th Australia New Zealand Intelligent Inf. Syst. Conf.*, Sydney, NSW, Australia, Dec. 2003, pp. 285-290.
- [65] E.D. Lughofer, "FLEXFIS: A robust incremental learning approach for evolving Takagi-Sugeno fuzzy models", *IEEE Transactions on Fuzzy Systems*, vol. 16, No. 6, pp. 1393-1410, 2008.
- [66] G.A. Carpenter and S. Grossberg, "Adaptive resonance theory (ART), in *The Handbook of Brain Theory and Neural Networks*", M.A. Arbib, Ed. Cambridge, MA: MIT Press, 1995, pp. 79-82.

Reactive-adaptive Methodology to encode Evolving Intelligent Agents in Serious Games

Mario Gongora

Centre for Computational Intelligence
De Montfort University
Leicester, UK
mgongora@dmu.ac.uk

David Irvine

Institute of Creative Technologies
De Montfort University
Leicester, UK
dirvine@dmu.ac.uk

Abstract—This paper describes our proposed methodology for the creation of novel AI based Agents capable of evolving to self-improve. We use AI concepts which are well established in the implementation in character behavior for games, such as Fuzzy Logic, the Belief-Desire-Intention model, and Finite State Machines; and discuss their limitations. We present a novel combination of these techniques to optimize the process of self adaptation for evolving intelligent agents.

Keywords: *Evolvable Agents, Agent Behaviour, Hybrid Systems, Self Developing Systems*

I. INTRODUCTION

In the areas of character simulation, such as serious gaming or populating virtual worlds, Intelligent Agents are a powerful tool to implement character behavior. Currently the implementation of these agents' intelligence relies on a primary AI technique, sometimes implemented using exclusively a single methodology. The mostly used techniques are Belief Desire Intention [1], Fuzzy Logic [2], or Neural Networks [3]. While these methods are adequate for basic character behavior in a prescribed environment (e.g. a standard game), using them to create complex Agents requires the particular methodology to be stretched to its limits by modifying it, creating a very complex implementation. In addition, if the agent is to be capable of self-adaption during its *life* (i.e. online execution), current methodologies pose additional restraints. We believe that a hybrid AI system with a well designed structure will result in a more efficient implementation for Agents that are required to behave in a semi-realistic manner and be capable of evolving during their *lifetime*.

In this paper we discuss current AI tools in the field of character behaviour simulation, their strengths and limitations. We present our hybrid system that uses a combination of AI methods, normally unsuitable for self adaptation, with a novel structure that makes it possible to apply constructive methods and enable an evolving Intelligent Agent.

This paper is organised as follows; Section II provides a background of the base techniques that we intend to combine; Section III presents our proposed hybrid complementary method, in Section IV we analyse our structural novelty, concluding in Section V and proposing further work in Section VI.

II. BACKGROUND

A number of well studied techniques within the field of AI are commonly used to implement the behaviour of semi-realistic Agents. On their own, these techniques often contain inherent limitations that make it complicated to achieve a fully featured behaviour simulation. The following sections analyse the main aspects of these techniques.

A. Belief Desire Intention

Belief/Desire/Intention (or BDI) is a method of modelling behaviour in rational agents. It can be seen as giving “mental attitudes of Belief, Desire and Intention, representing, respectively, the information, motivational, and deliberative states of the agent” [4]. The BDI model is based in the areas of both Philosophy and Cognitive Science, including work on Rational Agents [5]. Rational agents have bounded resources and usually incomplete knowledge about their environment. This incomplete knowledge is usually defined as the agents' belief network.

Knowledge is “that which is true in the world” while “beliefs are an agent's understanding about the world” [6]. Therefore agents' beliefs can be false or incorrect. Beliefs can also change over time as the agent receives more accurate information. Desires are what the agent wishes to accomplish in regards to its current objectives. These “objectives, or their priorities, are being generated instantaneously or functionally, and thus do not require any state representation” [4]. Desires are not necessarily consistent or compatible, and some cannot be achieved simultaneously. Goal deliberation over consistent desires results in the production of goals [7].

Plans describe the method with which agents achieve their goals and react to events, and can include sub goals. Other plans are executed to achieve these sub goals, creating a hierarchy of plans. When an agent has decided on a plan it commits itself to the plan, forming an intention [7].

BDI has been implemented in several research models, as well as applied in several situations such as Air Traffic management [4], military combat simulation [8], and has subsequently been implemented into frameworks such as the Procedural Reasoning System [9], Jadex [7] and JACK Intelligent Agents [10].

B. Fuzzy Logic

As noted in [11] among many others, fuzzy logic has been used in robotic behaviour systems as it provides a method that can cope with the uncertainty inherent to natural environments. Fuzzy logic has the capacity to represent uncertainty with fuzzy set theory [12]. Fuzzy sets are the most relevant to this work, as is their use that can make a behavior more realistic and adequate to handle real world scenarios. Fuzzy sets can also be powerful when dealing with linguistic variables. And language can be a primary interface between agents or between agents and human users of a system.

In agent behavior simulations, the main use of fuzzy sets is the implementation of sensorial input. In a simulated environment, agents have an artificially intrinsic capability of seeing the whole world and the complete set of variables that represent its environment, including what it would *detect* while exploring this environment. By using fuzzy sets, in combination with simpler variable hiding, a more realistic view of the world can be achieved. In [2] the authors combined fuzzy logic with BDI in the analysis of the surroundings, enabling the Agents to group data into *sets* and then use these sets to make decisions with their BDI systems. However the BDI required a number of explicit changes to deal with certain situations; “Commitment” to Intentions must be added to prevent an Agent from constantly changing its Intention, and Desires must be weighted to ensure more urgent Desires are met faster than less urgent Desires.

C. Finite State Machines

Finite State Machines are by far the most frequently used in today's computer games as a method for generating AI behaviour. In most cases, non-player characters (NPC) behaviour can be modelled in terms of mental states, and hence, FSMs are a natural choice when designing NPCs [13].

Each state that an Agent can enter into has to be foreseen and predefined by the programmer and then entered into the system. When the Agent enters this state it follows the rules laid down by the programmer. The Agent can enter an alternative state and follow another set of rules, or can remain in the same state if the rules fail to change it. If well thought out, this approach can lead to convincing AI that can carry out tasks with some degree of success, however, as the amount of states increases, so does the complexity of the FSM. For complex behaviour implementation, FSMs are not scalable.

Code complexity in a FSM increases significantly as the system scales up [14]. Iterating through all the state options may become too computationally expensive for a system that needs to run in real-time. By definition, a FSM is made up of a finite number of states; this number is implicitly fixed, as a variation at run-time would potentially violate the *finite* principle. Therefore if a state is not predefined by the programmer this cannot be handled or considered by the system at any time once deployed.

Still, FSMs are a powerful and efficient tool in the correct context, and therefore we aim to use them to deal with very specific states that we expect our agents to encounter, where it becomes necessary to control their actions directly.

III. PROPOSED HYBRID METHOD

The techniques presented briefly in the previous section are all capable of producing intelligent Agents, but it is often the case that a substantial amount of adjustment of these techniques is necessary to achieve all aspects of a required behavior, even if simplified. A BDI system often spends a high proportion of time in deliberation, and needs careful adjustments to ensure it has Commitment to an Intention, while preventing it from constantly changing which Intention it acts upon.

Finite state machines can become far too large for a complex AI simulation; and are intrinsically designed to be fixed for the entire *lifetime* of the agent. An attempt to insert or delete states at run-time can be catastrophic or it can become an unfeasible or impractical validation problem from the algorithmic point of view.

But by combining these two technologies in a very specific way, we formulate a methodology to enable the creation of complex semi-realistic agents more efficiently. In addition we can enable the structure to be adapted at run-time so that the Agent can evolve during its *lifetime*. In [14] the authors propose that research should be undertaken to determine if hybrid models of finite state machines and a BDI Agents' can reduce the design and coding while increasing the performance of a system. There appears to have been no subsequent reports of such research undertaken in depth. In this work we show how we can not only address this, but provide the necessary structure for an evolving system.

We take our inspiration from how many creatures, including human beings, behave; at our core, we have instincts, as all creatures do. We have then built upon these instincts through our evolution to create a method of reasoning for many other problems that require reasoning rather than reactions. When we take this type of behaviour and convert it to the closest matching AI technologies, we are presented with the following method:

Our Agents are primarily controlled by a BDI system; this is our emulation of human reasoning. The virtual world in which our Agents exist is simulated in a crisp fashion, therefore it is necessary to simulate the belief system of our Agents, adding uncertainty and imperfection to their beliefs; this will be addressed in the following section. We pass these beliefs on to the BDI system to enable our Agents to have a more realistic view of the virtual world. A FSM is given overriding control of the BDI system for situations where direct action must be taken with no deliberation, providing our emulation of instincts. In this way, we define a reaction-reasoning hierarchical system. Later we will show how we convert the reasoning system into an adaptive system which results in our reactive-adaptive methodology.

A. Belief Simulation

It is important to remember that an autonomous agent's decisions and plans are influenced by the information it receives about the world in which it is operating [8]. For the purposes of attempting to accurately simulate real-world behaviour, a purely computational agent is at an immediate

advantage; without the additional stage of filtering data that the agent receives from the Virtual Environment, the agent is automatically given a cognition of a situation that no real-world creature is capable of obtaining – our Agents automatically “see and understand everything”, and as such we must inhibit this ability.

Work on simulating belief systems of agents has already been undertaken by [8]. They suggest that when simulations are used for real-world decision support tools, it is important that the agent’s belief processes are also properly simulated, i.e. based on realistic observation and detection techniques.

The authors in [8] also suggest “conceptualizing the agent’s activities in terms of three main steps: detection, measurement and interpretation”. This idea has been carried over to the belief simulation in our work. As the Agent picks up (detects) new information it should be measured in such a way that the level of accuracy is inversely proportional to the distance.

Once a level of accuracy has been determined we are able to affect the quality of information that the agent receives. This can be defined as the “interpretation” stage. Given a set of imperfect data the agent must make a decision on how to interpret it. If an object is visible in the distance to the Agent, it is less likely to interpret it correctly as opposed to a nearby object. By determining how imperfect data it has received is, the Agent is capable of making a decision of what to do with the data. This decision making process is handled by our BDI implementation. As this is based on Fuzzy Systems then we can enable evolvable membership functions to adapt towards a realistic simulation; therefore there are no issues here.

B. Planning and decision making using BDI

Decision making via a BDI system enables us to control our Agent’s higher level functions, such as social interaction, complex problem solving, and other behaviours that are not otherwise covered by human instinct.

The authors in [4] proposed an abstract interpreter for BDI:

```
BDI-interpreter
Initialize-state();
repeat
    options := option-generator(event-queue);
    selected-options := deliberate(options);
    update-intentions(selected-options);
    execute();
    get-new-external-events();
    drop-successful-attitudes();
    drop-impossible-attitudes();
end repeat
```

This shows the basic interpreter cycle for a BDI agent. The agent updates its beliefs, and then updates its available options. Once it has done this it examines its desires, matches them against these options and proceeds to deliberate over them, before updating its intentions and executing the plans to facilitate those intentions. In the final steps, the interpreter removes successful and impossible goals and intentions.

By expanding on this abstract interpreter we are able to form a method that allows our Agents to make decisions about the virtual world based on imperfect values it receives, and then form plans to determine its actions in this virtual world.

Methods to expand the abstract interpreter already exist in several implementations, including JADE [7], METATEM [15] and AIL [16]. These existing implementations provide a software framework for the creation of goal-oriented agents [7], and as such could be used to implement BDI in our hybrid system.

BDI Agents can suffer in the aspect of realism due to their rational nature. If not carefully controlled, a rational Agent can, in some cases, continue a behaviour that could not be considered rational in real terms. In a situation where an Agent is trying to achieve a Desire with a high priority, it may ignore a situation which would naturally have a higher priority; thereby causing what appears to be irrational behaviour. While this issue can be overcome with a suitably well developed implementation, it can be avoided entirely by extracting behaviours from the BDI Agent that could be considered “priority behaviours”, or, “instinctual behaviours”. By handling these behaviours outside of BDI, we can avoid the potential problem of the Agent ignoring them.

C. Instinct emulation using FSMs

Instinct is often an overriding factor in behaviour. In situations that are life threatening, instinct comes into play and determines a behaviour to act to the situation with little reasoning or thought for consequences. While instincts can be controlled, it is more common (and expected) to let an instinct take hold and dictate the actions.

While a BDI system is capable of simulating instinct using pre-programmed Plans, due to the nature of BDI it is often costly to run this simulation with respect to the rational situations. When combined with Commitment to a Plan, BDI reasoning can often override instinct even in situations where instinct would more readily override decision making in the real world. This shortcoming can be mitigated by fine tuning of the BDI system and an Agent’s commitment to certain plans, but this is often time consuming. A FSM is more efficient at handling predefined behaviours, equivalent to instinct.

Finite State Machines are capable of making an Agent act a certain way in a finite amount of states with little processing requirement, therefore making them very fast when the list of states is small. The larger a FSM becomes, the less manageable it is for the systems programmer, and the more states the Agent must iterate through to determine its behaviour. For this reason, in our methodology, a FSM should only be responsible for the states that we define as being “instinctual”. Our structure will contain a small set of basic instincts, dealing with very specific cases; basically we will define a single *rational* state and a small set of *instinctual* states.

After the Agent receives data from the belief simulation, this is passed to the BDI system for interpretation. Once evaluated, the FSM is able to judge the state that the Agent is currently in. If the Agents’ state is recognised as “instinctual”, the FSM seizes control from the BDI. Once the Agent’s state

has changed to rational, the belief simulation system generates a new set of beliefs and acts upon them.

IV. ANALYSIS OF REACTION-ADAPTIVE METHOD

When analysing our hybrid FSM-BDI system we see that by combining these methods in the correct way we can produce a technique which is more efficient computationally, as well as more effective in simulating intelligent behavior and enables the system to evolve during its life.

We will define a test scenario in which to place a group of Agents to illustrate and evaluate our novel methodology.

A. Test Scenario

For this analysis, a number of Agents will be placed at random in a virtual environment; each of these Agents has a basic need for social interaction. In addition to this basic need, they follow the instinctual desire to avoid outbreaks of fire. Each Agent is given 4 traits (Wealth, Education, Authority and Fitness), which will each be assigned a value between 0 and 1.

At this stage we will simplify the Agents' traits to this [0-1] range rather than formal fuzzy sets as this is a known technique we have borrowed from other research and does not form part of our novel approach. In actual implementations we can formalize this fuzziness and the evolving capabilities will remain the same in principle.

By varying the degree of *simplified-membership* of each trait that every Agent has, we are able to create Agents that have different degrees of compatibility with each other. When created, each of the Agents is free to move about within the virtual environment at a certain speed. As the Agents move they look in front of them with a limited visual radius and focal length. Any other Agents that they encounter within this visual range they should approach and "interact" with. This interaction allows for a calculation of how compatible each Agent is with each other. This is a simple calculation based on comparing the degree of each trait each Agent has:

$$\text{Compatibility} = \begin{cases} 1 & \text{if } \delta\text{Traits} \leq 1 \\ 0 & \text{if } \delta\text{Traits} > 1 \end{cases} \quad (1)$$

where:

$$\begin{aligned} \delta\text{Wealth} &= \text{Agent}_1.\text{Wealth} - \text{Agent}_2.\text{Wealth} \\ \delta\text{Education} &= \text{Agent}_1.\text{Education} - \text{Agent}_2.\text{Education} \\ \delta\text{Authority} &= \text{Agent}_1.\text{Authority} - \text{Agent}_2.\text{Authority} \\ \delta\text{Fitness} &= \text{Agent}_1.\text{Fitness} - \text{Agent}_2.\text{Fitness} \\ \delta\text{Traits} &= \delta\text{Wealth} + \delta\text{Education} + \delta\text{Authority} + \delta\text{Fitness} \end{aligned}$$

If the other Agent is "compatible", this Agent remains stationary as it has completed its task of finding a compatible Agent. If the other Agent is not compatible, this Agent moves off in search of other Agents.

At randomly spaced intervals a fire breaks out at a random location in the virtual world. Any Agent near to the fire should

then move away from it. Once they have moved away they return to their default searching behavior.

B. Modelling of Test Scenario

As an initial test, we can model the different behaviour in a classical way using each one of the FSM or BDI techniques.

1) FSM Based Model.

This model demonstrates all the possible states that our Agent can enter in our test scenario, as well as determining the transitions that can occur to move the Agent between states. In every state it is possible to transition to the "Escaping Fire" state, and it should be noted that the addition of any further states would cause the model to become increasingly complex, as those states would also need to transition to "Escaping Fire". It is important to consider as well that we may not have included all the possible states that the Agent should be able to enter; without repeated tests of the scenario, not all possibilities may have presented themselves. Fig. 1 shows such a basic static behavioural model.

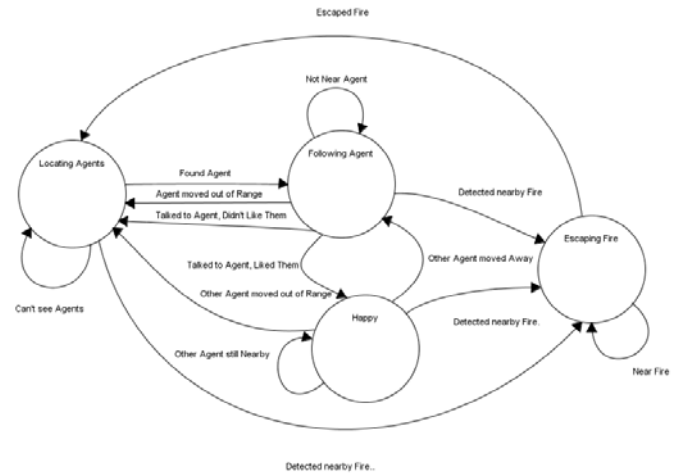


Figure 1. FSM model for Implementing our test scenario.

2) BDI Based Model.

To implement our test scenario using BDI the Agent would need to have the following BDI components:

Beliefs

- Location of other Agents
- Compatibility with other Agents
- Location of Fire
 - Proximity to Fire

Desires

- Socialise with others
- Avoid Fire

Intentions

- Move to another Agent
- Talk to/determine compatibility with other Agents
- Move away from Fire

This model does not deal with the complexities of adding values of “Commitment” to each Intention. It should also be noted that the Intentions listed here are conceptual in basis, as a BDI Agent only has one Intention at any time, selected from its Desires. We also do not deal with the priority of any Desire; without this an Agent can follow a Desire blindly when another of its Desires should clearly take precedent.

While this model is not complex, implementing it highlights at least one of the potential limitations of such a system; namely that the computational cost of each BDI Agent is significantly higher than for a FSM implementation [14].

3) Self-adaptation.

From the FSM model presented in 1) we can see that adapting to an additional state becomes a major problem. Even if the ethos of FSM is ignored, the inclusion of an additional state becomes a matter of exhaustively analysing the relation and transitions between all states with each other.

This presents two main problems for an evolvable system:

- Scalability becomes a serious issue and with every new *attempt* to adapt, the computational load increases exponentially
- Analysis of relations and transitions, as this is a very complex process, normally done by a human designer. It would require a whole *Intelligent Analysis tool* to pursue this task!

The BDI model presented in 2) presents its own set of limitations in this respect. On the positive side, as the evaluation of every layer of B, D and I are sequential, real-time adaptation increases the computational cost only linearly; even if the computational cost itself is heavier than that for FSMs. This is a clear advantage over FSM.

On the other hand, as every component of a layer is a parallel condition, and independent of the condition in the other layers, evolving to a new condition does not require a very thorough analysis of the holistic interaction of every component... so far an additional advantage over FSM.

But this is only valid as long as the prioritization relationships between behaviours, and between desires relative to behaviours, and between intentions relative to desires are ALL non-critical. If this is not the case (as it would always be in a BDI only implementation), then there is a serious complexity issue in terms of analysing these relationships at an online adaptation stage.

Therefore, with any of the alternatives in 1) and 2) we have not only scalability and performance issues, but it becomes impractical to enable an evolvable system.

4) Proposed Evolvable Reactive-Adaptive Model.

Our hybrid model will combine the most appropriate components from both FSM and BDI in order to reduce the overall complexity of the system as well as reducing the computational cost of running the test. In addition, if the structure is carefully formed, we can enable as well an evolvable system. Fig. 2 shows the FSM component of the same model presented in 1) and 2) earlier in this section.

This FSM component deals with situations where we believe an “instinctual behaviour” is necessary to resolve the situation. In this case the need for an Agent to avoid nearby fires has been defined in this category. The “Escaping Fire” is an instinctual state; the “Not Escaping” is *the* rational state. At this point it is important to clarify that there can be more than one instinctual state, while these all gravitate around a single rational state, therefore *the* rational state could be just called that.

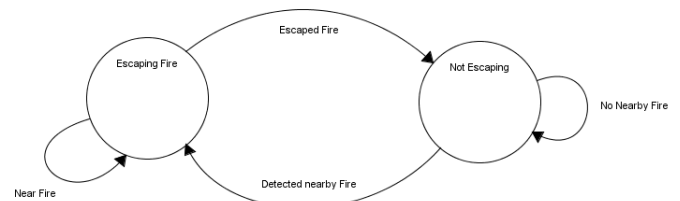


Figure 2. FSM model for Implementing our hybrid test scenario.

Following from the rational state, in this model, the BDI system handles any other situations when the FSM is in the “Not Escaping” state. At this point, ALL the BDI components fall into the rational category and their relations are non-critical across all the layers.

Beliefs

- Location of other Agents
- Compatibility with other Agents

Desires

- Socialise with other Agents

Intentions

- Move to other Agents
- Talk to/determine compatibility with other Agents

C. Analysis of Hybrid Model Effectiveness

Runtime performance for a FSM system compares favourably to that of a BDI system, especially when the amount of Agents is significant within the simulation environment. Our BDI model maintains a low complexity when compared to the FSM design, and adding more behaviours increases the complexity in a linear fashion [14]. However, it is a complex task to include information regarding the Agents commitment values or its priority values for plans. Without this planning, Agents using the BDI model are subject to situations where they react in an unrealistic manner. The inclusion of this planning, apart from being unreliable, makes the adaptation a prohibitively complex problem.

In our combined model, the FSM method becomes significantly less complex, requiring only the minimum number of states to handle “instinctive behaviour”; in this case we have only 2 states from the original 4. More critically, the number of transitions, where the complexity grows exponentially, has been reduced from 13 to 4. When the FSM is in its rational state, it passes control to the BDI system, which means only one of the models is effectively active at any one time. The BDI model itself is also less complex as it has fewer Desires; we have in total 5 out of the original 9 elements. By keeping control of instinctive behaviours in the FSM, we

are able to avoid the higher computation cost of a BDI Agent and react more quickly to situations where it is deemed necessary, without the requirement of deliberating over a BDI system's Desires and checking commitment and priority values.

Moreover, as the BDI system now has absolutely no need to consider priority values for plans, the task of evolving the system for online adaptation becomes feasible using established constructive techniques. The authors have applied these principles to other problems [17], [18] and many others explored by the research community.

The BDI system in our Reactive-Adaptive model has been relieved of the following constraints:

- There are no prioritization relationships between behaviours as the FSM takes over this task at an external point to the entire BDI system
- There are no prioritization relationships between desires relative to behaviours as these are now always independent with no need for a hierarchical structure
- The prioritization relationships between intentions relative to desires are non-critical as all the desired have equal validity and have no hierarchies to follow from the behaviour level.

These characteristics make our BDI section of the structure consistent with a single independent layer of rules, which can now enable an evolvable system as it can support a *flexible* rule based model [19] or an online reward feedback function [17]. These techniques can now be applied without invalidating the decision making process; and without requiring an exhaustive consistency check every time an automatic adaptation occurs.

V. CONCLUSIONS

While existing AI methodologies are capable of creating Agents that behave in a semi-realistic manner, these techniques are often very complex to implement or result in behaviours that can be clearly unrealistic. In addition, their structure and limitations make it impractical to activate a reasonable or effective evolvable adaptation.

We have shown how using a novel hybrid and hierarchical combination of these methods we created a technique that brings the best features of existing and proven methodologies while attempting to avoid their limitations. By analysing these widely used AI methodologies in our test scenario we have been able to present a case that demonstrates how our combination of techniques creates a simpler implementation that results in a more efficient and realistic simulation than using techniques individually.

The main achievement is that a coherent structure suitable for self-adaptation can be enabled using techniques that normally have serious intrinsic limitations to achieve this.

VI. FURTHER WORK

For the purpose of testing both the computational cost and realism of behaviour produced, it will be necessary in the

future to implement our hybrid technique in a more complex scenario while providing an automated measure of realism.

REFERENCES

- [1] Patel, Purvag and Hexmoor, Henry. Baltimore, "Designing BOTs with BDI agents." International Symposium on Collaborative Technologies and Systems, 2009. pp. 180-186.
- [2] Li, Yifan, Musilek, Petr and Wyard-Scott, L.. "Fuzzy logic in agent-based game design." Proceedings of the 2004 Annual Meeting of the North American Fuzzy Information Processing Society. pp. 734-739.
- [3] Rhalibi, Abdennour El and Merabti, Madjid. "A Hybrid Fuzzy ANN System for Agent Adaptation in a First Person Shooter." International Journal of Computer Games Technology, Volume 2008 (2008), Article ID 432365, 18 pages, doi:10.1155/2008/432365.
- [4] Rao, Anand S and Georgeff, P Michael, "BDI Agents: From Theory to Practice." Proceedings of the First International Conference on Multi-Agent Systems, San Francisco, 1995.
- [5] Bratman, Michael E. "Intentions, Plans, and Practical Reason", Massachusetts : Harvard University Press, 1987.
- [6] Genesereth, M R and Nilsson, N J. "Logical foundations of artificial intelligence", Los Altos : Morgan Kaufman Publishers, Inc., 1987.
- [7] Braubach, L., Pokahr, A. and Lamersdorf, W. "Jadex: A BDI Reasoning Engine", Multi-Agent Programming Volume 15. s.l. : Springer US, 2005, pp. 149-174.
- [8] Bhargava, Hemant K and Branley, Jr., William C. "Simulating belief systems of autonomous agents" 1995, Decision Support Systems 14, pp. 329-348.
- [9] Ingrand, Francois F, Georgeff, Michael P and Rao, Anand S. "An architecture for Real-Time Reasoning and System Control", 1992, IEEE Expert: Intelligent Systems and their Applications, 1992 pp. 34 – 44.
- [10] Busetta, Paolo, P Busetta, R Ronnquist, A Hodgson, A Lucas, "JACK - Components for intelligent agents in JAVA", AgentLink News Letter. 2 Jan 1999, pp. 2-5.
- [11] Saffiotti, A. "The uses of fuzzy logic in autonomous robot navigation", 1997, Soft Computing - A Fusion of Foundations, Methodologies and Applications, pp. 180-197.
- [12] Zadeh, L A and Bellman, R E. "Decision-Making in a Fuzzy Environment" 1970, Management Science, pp. 141-164.
- [13] Yue, Billy and de-Byl, Penny., "The state of the art in game AI standardisation", Proceedings of the 2006 international conference on Game research and development, Perth : Murdoch University, 2006, pp. 41-46.
- [14] Bartish, Arran and Thevathayan, Charles. "BDI Agents for Game Development", Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2. ACM, Bologna, Italy, 2002, pp. 668 – 669.
- [15] Fisher, Michael. Heidelberg, "A survey of concurrent MetateM — The language and its applications", Temporal Logic—Proceedings of the First International Conference (LNAI Volume 827). Springer-Verlag, 1994, pp. 480-505.
- [16] Dennis, L A., Farwer, B., Bordini, R. H., & Fisher, M., "A Common Semantic Basis for BDI Languages", . In *Programming Multi-Agent Systems*. Springer Berlin / Heidelberg, Springer, 2008, (pp. 124-139).
- [17] A. Huemer, D. Elizondo, M. Gongora, "A Reward-Value Based Constructive Method for the Autonomous Creation of Machine Controllers", Lecture Notes in Computer Science, Vera Kurkova, Roman Neruda and Jan Koutnik Eds., Vol. 5164, 2008, pp. 773-782.
- [18] M. Gongora, M. Rodas, "Analysis of Organized Asymmetry Development Using Artificial Cellular Differentiation Models", Proceedings of the 2009 IEEE Workshop on Evolving and Self-Developing Intelligent Systems (ESDIS 2009), Nashville, TN, USA, April 2009.
- [19] Angelov, P. P. "Evolving rule-based models: a tool for design of flexible adaptive systems", Springer, 2002, ISBN: 978-3-7908-1457-6.

Lifelong Learning by Evolution in Robotics: Bridging the Gap from Theory to Reality

B. Santos-Diez, F. Bellas, *Member, IEEE*, A. Faiña, R.J. Duro, *Senior Member, IEEE*

Abstract—In this paper, we describe the practical computational implementation of the Multilevel Darwinist Brain (MDB) cognitive architecture. The MDB follows a developmental robotics approach, where open-ended, autonomous learning systems that continually adapt to their environment are designed. Consequently, the acquisition of knowledge, represented by models, required to select robot actions is obtained through the interaction with the environment. The computational implementation of this type of architecture implies several aspects that are very relevant: evolutionary algorithms that must support a real time operation architecture, processes running in different time scales, memory elements that are continuously interplaying, and so on. In this work, we will focus on these aspects, describing the current implementation of the MDB and its successful performance in a real robotics learning scenario.

I. INTRODUCTION

The application of evolutionary techniques in autonomous robotics has been a very prolific topic in the last twenty years [1]. But most of these approaches, instead of designing architectures that provide robots with real autonomy, were focused on designing controllers for particular tasks, where the acquisition of knowledge was very limited, mainly obtained in a controlled fashion, and the problems of adaptation to the dynamics of the real environments were simplified [2][3][4].

Nowadays, we know that a control system such as the one that is necessary for realistic autonomy is something that goes beyond traditional control in terms of the specifications or requirements. These additional requirements imply the ability to learn the control function from scratch, the ability to change or adapt it to new circumstances, the ability to interact with the world in real time while performing the aforementioned processes and, in some instances, even to change the objectives that guide the control system. As in the case of animals, open-ended lifelong learning systems present the potential of achieving a more realistic level of autonomy.

Starting from this background, in the early 2000's Weng proposed an autonomous mental development (AMD) [5][6] line, particularized to robotics through the cognitive developmental robotics (CDR) [7] approach. The key aspect of CDR is that the control structure should reflect the robot's

own process of understanding through lifelong interactions with the environment. We have followed this CDR approach in the design of the Multilevel Darwinist Brain cognitive architecture, but we have addressed the problem making use of some of the concepts of traditional cognition, introducing ontogenetic evolutionary processes for the on-line adaptation of the knowledge bearing structures.

Therefore, our approach has been to obtain a developmental control architecture for real robots that uses evolutionary algorithms in real time. In this line, the number of examples found in literature is very small. We can point out [8], where the authors present an approach called “embodied evolution” in which a group of robots can improve in real time a basic set of behaviors through evolution and use a system whereby they mate to transmit genetic information. In [9][10] the authors resort to competitive coevolution to make two physical robots adapt their behaviors in real time.

The main drawback of evolutionary approaches is their computational cost. Therefore, the computational aspects of the implementation must be carefully analyzed and the main objective of this paper is to discuss these aspects within the Multilevel Darwinist Brain (MDB) cognitive architecture. We will first describe the architecture from a theoretical point of view to better understand all the implementation decisions that have been taken, and that have resulted in successful performance in real robotics learning tasks.

II. COGNITIVE ARCHITECTURE FOR LIFELONG LEARNING

Cognition may be defined as “The mental process of knowing, including aspects such as awareness, perception, reasoning, and judgment”. Designing a cognitive architecture for a robot requires dividing the problem into simple elements that can be studied separately. On one hand, we have the mental process of knowing, that is, in more mathematical terms, of extracting models from data can be employed in the process of decision making so that appropriate actions may be taken as a function of sensing and motivation. On the other, we have the decision making process itself, and, in robotics, a decision is always related to an action or sequence of actions. In a sense, the models must be used in order to decide the appropriate actions so as to fulfill the robot's motivations. It is in how the model making and the action determination processes take place that cognitive architectures differ from each other.

B. Santos-Diez, F. Bellas, A. Faiña, and R.J. Duro are with the Integrated Group for Engineering Research in the University of Coruna, Spain (e-mail: borja.santosdiez@udc.es, fran@udc.es, afaina@udc.es, richard@udc.es).

A. A little formalism

The particular cognitive model that formalizes the previous ideas starts from the premise that, to carry out any task, a *motivation* (defined as the need or desire that makes an robot act) must exist that guides its behavior. Therefore, the *satisfaction* of the robot can be defined as a magnitude or vector that represents the degree of fulfilment of the motivation or motivations of the robot. The final objective of the cognitive architecture is to explore the possible action space in order to maximize the resulting satisfaction.

To obtain a system that can be applied in reality implies internally (without interaction with the environment) deciding on the best action to apply within a time span that permits real time interaction with the environment. To internally evaluate candidate actions some theoretical functions are required that must be somehow obtained. These functions correspond to what are traditionally called:

- *World model (W)*: function that relates the *external perceptions* before and after applying an action.
- *Internal model (I)*: function that relates the *internal perceptions* before and after applying an action.
- *Satisfaction model (S)*: function that provides a predicted *satisfaction* from predicted perceptions provided by the World and Internal models.

As commented before, the main starting point in the design of a developmental cognitive architecture was that the acquisition of knowledge should be automatic, so we establish that the three models *W*, *I* and *S* must be obtained during execution time as the robot interacts with the world. To perform this modelling process, information can be extracted from the real data the robot has after each interaction with the environment. Hereafter, these data will be called *action-perception pairs* and are made up of the sensorial data in instant *t*, the action applied in instant *t*, the sensorial data in instant *t+1* and the satisfaction in *t+1*.

Summarizing, for every interaction of the robot with its environment, functions *W*, *I* and *S* must be modelled using the information in the action-perception pairs and the best action for the robot selected using the models available at that time and maximizing predicted satisfaction.

To create models is to try to minimize the difference between the reality that is being modeled and the predictions provided by the model. Consequently, a cognitive architecture must involve some type of optimization strategy or algorithm. As the search spaces are not really known beforehand and are usually very complex, the MDB is based on using one of the most powerful stochastic multipoint optimization techniques: artificial evolution.

B. The Multilevel Darwinist Brain architecture

The Multilevel Darwinist Brain (MDB) is a general Cognitive Architecture first presented in [11] that permits an automatic acquisition of knowledge in a real robot through the interaction with its environment, making it capable of adapting its behaviour to the context and objectives. The

inspiration for the approach followed comes from the classical bio-psychological theories by Changeaux [12], Conrad [13] and Edelman [14] in the field of cognitive science relating the brain and its operation through a Darwinist process. All of these theories lead to the same concept of cognitive structure based on the brain adapting its neural connections in real time through evolutionary or selectionist processes. That is, the Multilevel Darwinist Brain uses *artificial evolution* as the tool for incorporating model learning.

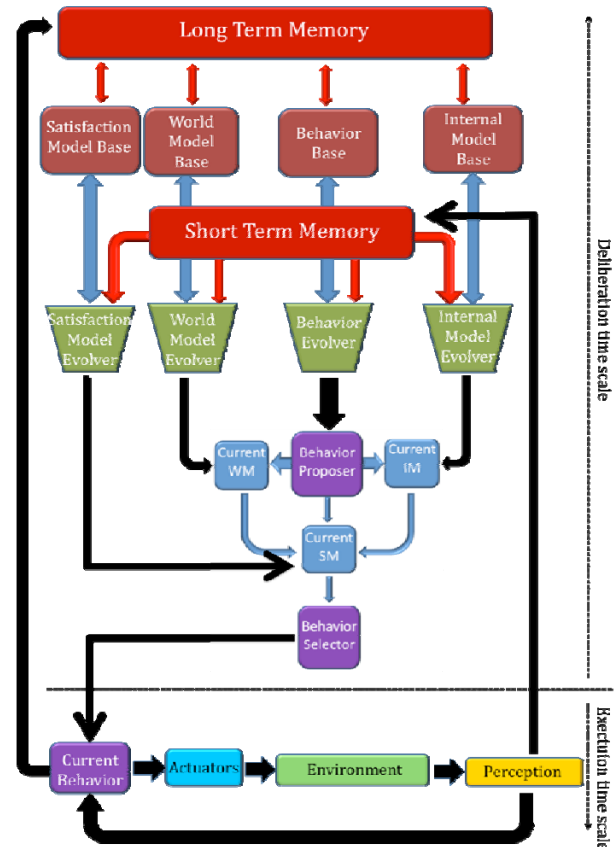


Fig. 1. MDB block diagram

Fig. 1 displays a block diagram of the MDB. It follows the previous cognitive model, which has been generalized by basically adding two new aspects: *behavior structures* that generalize the concept of action and *memory elements*, basically a short-term and a long-term memory required in the learning processes.

As shown in Fig. 1, the MDB is structured in two different time scales, one devoted to the execution of the actions in the environment (reactive part) and the other devoted to the learning of the models and behaviors (deliberative part). We can describe the operation of the MDB in these two scales:

1. *Execution time scale (reactive)*: which continuously repeats these two steps in a sequential manner:

- There is a *current behavior*, selected in the

deliberative process, which provides the action the robot must apply in a given instant of time as a function of the perceptual values it has as inputs.

- The selected action is applied to the *environment* through the *actuators* of the robot obtaining a new *perception*, that is, new sensing values.

2. *Deliberation time scale*: the processes included here can also be taken as operating concurrently and in different time scales, thus, *they are not sequential*:

- The acting and sensing values obtained after the execution of an action in the environment in the execution time scale provide a new action-perception pair that is stored in a *Short-Term Memory (STM)*.
- The evolutionary model learning processes (for world, internal and satisfaction models) try to find functions that generalize the real samples stored in the STM. Each evolutionary process has been represented by two blocks in Fig. 1, one representing the evolution itself (labelled *evolver*) and the other one representing the population for each evolution (labelled *base*).

Each evolutionary process will be continuously running and producing new models in their own time depending on the complexity of the models required and their practical implementation will require avoiding incoherences during the interplay of the elements involved in the real time operation.

- The best models in a given instant of time are taken as *current world, internal and satisfaction models* and are used by the *behavior evolver* to select the best behavior (the one that produces the highest predicted satisfaction).

The blocks labelled Current WM, Current IM, Current SM and Behavior Proposer are really included in the individual's evaluation stage within the behavior evolver.

- The behavior evolver is continuously proposing new behaviors that are better adapted to the STM contents. Upon request, the behavior selector provides the best one to the reactive part of the MDB, the *current behavior* block in Fig. 1. This behaviour should be better adapted to the current STM and, consequently, to the current "reality" of the robot.
- The block labelled *Long-Term Memory (LTM)* in Fig. 1 stores those models and behaviors that have provided successful and stable results in their application to a given task in order to be reused as seeds in the population for new learning processes.

Each interaction of the robot with the environment (*iteration*) is taken as the basic time unit within the MDB. As more iterations take place, the MDB acquires more information from the real environment and thus the model learning processes have more information available and the models, and consequently the behaviors obtained from them, are more reliable, leading to more appropriate robot actions.

C. Lifelong learning by evolution

The main difference of the MDB with respect to other architectures lays in the way the process of modelling functions W, I and S is carried out. Evolutionary Algorithms are the basis of this architecture and as representation for the models we have selected Artificial Neural Networks. Thus, the acquisition of knowledge in the MDB is basically a neuroevolutionary process.

The modelling here is not an optimization process but a learning process taking into account that we seek the best generalization for all times, or, at least, an extended period of time, which is different from minimizing an error function in a given instant t . Consequently, the modelling technique selected must allow for gradual application, as the information is known progressively and in real time. Evolutionary techniques permit gradual learning by controlling the number of generations of evolution for a given content of the STM. Thus, if evolutions last just a few generations per iteration (interactions with the environment), gradual learning by all the individuals is achieved.

To obtain general modelling properties in the MDB, the population of the evolutionary algorithms must be preserved between iterations, leading to a sort of inertia learning effect where what is being learnt are not the particular contents of the STM in a given instant of time, but of sets of STMs that were previously seen. In addition, the dynamics of the real environments where the MDB will be applied imply that the architecture must be intrinsically adaptive. This strategy of evolving for a few generations and preserving populations between iterations permits a quick adaptation of models to the dynamics of the environment, as we have a collection of possible solutions in the populations that can be easily adapted to the new situation.

In the case of behaviors, in the current version of the MDB they are also represented by ANNs, and consequently, they can be viewed as neural controllers that provide the action the robot must apply in the environment according to the sensorial inputs.

D. Learning from memories

The management of the Short Term Memory is critical in the real time learning processes within the MDB because the quality of the learned models depends on what is stored in this memory and the way it changes. The data stored in the STM are acquired in real time as the system interacts with the environment and, obviously, it is not practical or even useful, to store all the samples acquired in the robot's lifetime. A dynamic replacement strategy was designed that labels the samples using four basic features (distance, complexity, initial relevance and time) related to the saliency of the data and temporal relevance.

The Long Term Memory is a higher level memory element. From a psychological point of view, the LTM stores the knowledge acquired by the robot during its lifetime. This knowledge is represented in the MDB as

models (world, internal and satisfaction models), behaviors, and their context. The models and behaviors stored in the LTM in a given instant of time are introduced in the evolving populations as seeds so that if the robot returns to a previously learnt situation (or a similar one), the model will be present in the population and the prediction will be accurate soon. For a detailed description of the operation of the STM and LTM and the interplay between them see [15].

III. COMPUTATIONAL IMPLEMENTATION

The computational implementation of the concepts presented above is the key to their applicability in real robotic systems. In what follows we are going to consider some of the main elements in the current JAVA implementation of the MDB.

We can distinguish 4 basic packages that make up the computational core of the MDB: robot, evolutionary algorithm, model and memory.

1) Robot

As a first design requirement, already in the initial versions of the MDB, we imposed that the basic onboard processor of the real robots (usually a microcontroller) should be used just to execute actions and to collect the sensorial information in real time. Regarding the two different time scales shown in Fig. 1, this means that the onboard processor is in charge of the execution time scale elements. Thus, the deliberative part of the MDB is always executed in a separate processor (on or off the robot), and the communications between these two structures are carried out using the standard TCP/IP protocol.

The second basic design requirement related with the robot is that the MDB should be as independent from the particular hardware as possible. That is, we assume that the MDB receives sensorial information and provides actions to be applied in a robot, but its core cannot include any particularity about the robot. To this end, we have taken inspiration from the Player/Stage project [16], which uses a network server for robot control that provides an interface to the robot's sensors and actuators over the IP network.

The robot package includes all the classes implementing the previous two requirements. The designer must create a simple configuration file including a description of the robot sensors and actuators and the IP port where it is connected. On the other hand, for each particular robot or simulator, an interface program must be developed to provide sensorial information and to capture the actions obtained by the MDB using the standard TCP/IP protocol. The onboard processor's computational load is minimized and, consequently, these data are always in raw format. If any kind of processing is required, it will be carried out by the perception package.

2) Evolutionary Algorithm

This package includes all the classes in charge of executing the evolutionary processes for the models and behaviors, which constitute the core of the architecture. One

of the main drawbacks of the application of evolutionary algorithms in real robotics is the computational cost they imply, that makes them apparently unsuitable for real time operation. We have addressed this problem through the following design decisions:

- First, as commented in the MDB description, the evolution of the models and behaviors only last a few generations per iteration in order to obtain a smooth learning curve. This obviously reduces the computational cost in between interactions with the world and makes its real time implementation feasible.
- In addition, the MDB is intrinsically concurrent and each evolutionary process is automatically assigned to a different processor when available: each evolutionary process is an independent thread forcing the use of a different processor. In the case of having a local area network, the threads may be executed in different computers over the network automatically.

The behavior evolution module uses the current models to evaluate candidate behaviors. Taking into account the distributed execution of each model evolution, we have implemented a coherence protocol that ensures an updated evaluation of the individuals, using always the most recent models. A similar procedure has been implemented for the update of the current behavior in the reactive part of the MDB, which is replaced every time a better one is obtained but ensuring a coherent action selection.

On the other hand, the MDB principles establish that it is independent from the particular type of evolutionary algorithm and artificial neural network. This is very easy to support with the object-oriented design of the MDB. In fact, we have tested the architecture with the different algorithms implemented in the JEA library [17], by simply changing the class in the configuration file.

3) Model

To create a new experiment using the MDB, the first step is to set up the model configuration. This implies formally describing the world, internal and satisfaction models, that is, we must decide the knowledge representation within the architecture. This selection is very relevant because the success or failure of learning depends highly on the complexity of the models.

What the designer must do is simply indicate the inputs and outputs corresponding to each model. This information corresponds to the external and internal sensorial information and to the action space. We must take into account that the robot configuration file already includes all the real sensors and actuators, but here, as inputs and outputs to the models, we can use virtual sensors and actuators that process the raw data provided by the robot allowing the designer to use the sensorial information freely.

To automate the model configuration, we have developed a general procedure based on the division of independent sensorial information into different models. For example, in the case of the world models, if we have a robot with 4

infrared sensors and 2 light sensors, the MDB will create 2 models as a first approach. The first one relating the 4 infrared inputs in instant t and $t+1$ and the action applied, and the second one relating the 2 light inputs with the action in the same way. This procedure has provided the best results in several experiments [18][19].

What is relevant for the computational implementation is that we could have several concurrent evolutionary processes for all the submodels making up the world, internal and satisfaction models as well as for the behaviors. As commented above, a mechanism has been implemented that automatically executes these concurrent processes in different, and even remote, threads.

4) Memory

The evolutionary processes in the MDB are continuously trying to model the STM action-perception pairs, which represent all the sensorial and acting information for each iteration of the architecture. As commented in section II, the replacement strategy of the STM determines the type of learning achieved, and it must be adjusted depending on the complexity of the model. Taking into account the previously explained subdivision into models, using a single STM for all the models is not feasible.

Hence, in the current implementation of the MDB, each model evolution has its own STM with its particular replacement strategy. All the classes required for this management are included in the memory package. The corresponding configuration file only requires establishing the type of replacement strategy, while the creation and execution of the STM is automatic.

Regarding the LTM, the design and implementation follows basically the same principles: there is a different LTM for each evolutionary process, which is executed concurrently with it.

To summarize this section, we want to highlight the following implementation aspects within the MDB, all related with the operation of the architecture in real robots:

- Object-oriented design and JAVA implementation.
- Hardware/simulator independence through the use of a TCP/IP middleware approach.
- Time scale independence: reactive elements are executed onboard and deliberative elements in remote or onboard computers through TCP/IP communication.
- Automatic concurrent execution of the evolutionary processes in remote processors.
- Automatic division of STM and LTM memories and execution according to the evolutionary processes.
- Persistent evolutionary processes that never stop although the fitness function can change.
- Easy integration with evolutionary and ANN libraries

IV. AN ILLUSTRATIVE EXAMPLE

The MDB architecture has been widely tested in real examples [18][19]. But here we want to illustrate the operation of the MDB implementation in terms of real

operation. To this end, in this section we describe a real robot learning experiment that involves the Sony AIBO robot and a pink ball (Fig. 2). The goal is simple: the robot must learn to catch the ball as quickly as possible from its interaction with the environment through its sensors and actuators. To this end, we have followed a developmental approach where a teacher places the ball in the robot's surroundings. In the first iterations, the robot will lose the ball, so the teacher must place it again near it. Once the task is learned, the ball position is not so important because the robot has acquired the general skills.



Fig. 2. Behavior of the robot in the initial (left) and final (right) iterations of the experiment

Regarding the robot configuration, we are using camera data and head angle as sensorial information and a predefined gait for the actions the robot can execute, in this case, just moving. An IP port was assigned to the robot, and the MDB was run in a remote computer receiving the sensorial information and sending the actions through the network.

In terms of the models, as usual, they do not use the camera information directly, but processed in the perception package classes. The head angle is used directly and the gait of the robot is specified using linear and angular speeds. In particular, the information we are managing in the models is:

- *Distance to the ball*: calculated from the camera data.
- *Angle to the ball*: calculated from the camera data and the head angle.
- *Angular speed*: linear speed is fixed and the MDB must provide the angular speed required to reach the ball.

Following the previously commented procedure, we have created two different world models according to the different sensorial information they manage: one for distance prediction and another for angle prediction. In this case, the models will be represented by multilayer perceptrons, one with 3 inputs (distance, angle and angular speed in time t) and 1 output (predicted distance), and another one with the same 3 inputs and 1 output (predicted angle). The satisfaction model is another ANN with 2 inputs (predicted distance and angle) and one output (predicted satisfaction). As commented before, the goal for the robot is to catch the ball as soon as possible, so we have established that its motivation is to minimize the distance and angle (reaching the ball frontally) in the minimum number of iterations. As a consequence, the satisfaction value is simply the difference with respect to this motivation. In this example, we did not use internal models (see [19] for an example with them).

With this setup we have configured 3 concurrent evolutionary processes that were run in an Intel Core 2 Duo processor, using 3 concurrent threads. We have used the JEAF library and, in particular, the Differential Evolution evolutionary algorithm for the 3 cases because it provided the best results with a low population size. The STM was implemented using a purely temporal strategy (FIFO).

In this case, the behaviors are simple actions, so the behavior evolution is not required. The MDB operation is exactly the same, but instead of using an evolutionary process to obtain behaviors, we use a simple heuristic to select the action for each instant of time.

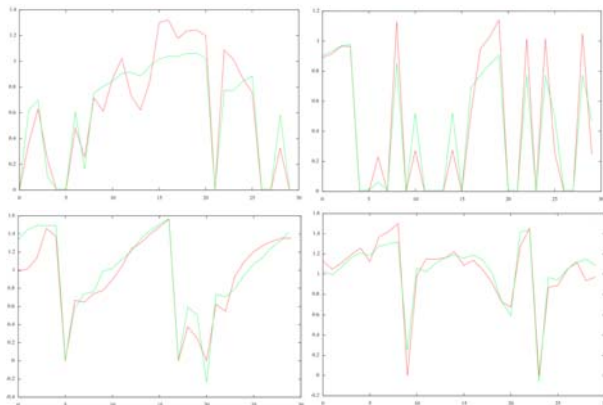


Fig. 3. Modelling (dark line) provided by the best satisfaction model with respect to the STM contents (light line). The x-axis corresponds to the number of sample in the STM and the y-axis to the satisfaction value. Top left graph corresponds to iteration 5, top right to iteration 15, bottom left to iteration 25 and bottom right to iteration 30

With all these elements, the experiment execution was successful requiring, on average, 30 iterations of the robot to learn the task. This implies around 25 minutes of teacher real time interaction, which is an affordable time for this task. Fig. 2 shows a typical trajectory of the robot in the initial iterations (left) where the models are still poor and in iteration 30 (right) where the models have been improved and the robot reaches the ball successfully.

Fig. 3 displays the modeling provided by the best satisfaction model with respect to STM content in four separate iterations. This behavior is the same for the other two models and, as we can see the MDB, works as expected obtaining a gradual learning that improves with time. Furthermore, the case shown in this figure is interesting because it illustrates the potential of using a satisfaction model to guide robot behavior. If in a given instant of time the motivation changes, the only implication it has is that the satisfaction model will fail in its prediction of the satisfaction, and will eventually adapt to the new situation, without affecting the world and internal models. This feature provides real autonomy to the robot as it allows for changing motivations through the satisfaction model and changing environments or even robot structure through world and internal models.

V. CONCLUSION

We have analyzed the computational implementation of the Multilevel Darwinist Brain in order to demonstrate how an evolutionary algorithm can allow for real time robot operation if it is adequately implemented by separating the reactive and the thinking parts, and allowing them to operate concurrently on different time scales providing the appropriate mechanisms for their coherent operation.

ACKNOWLEDGMENT

This work was supported by Xunta de Galicia (09DPI012166PR) and European Reg. Development Funds.

REFERENCES

- [1] Walker, J, Garrett, A., Wilson, M. "Evolving Controllers for Real Robots: A Survey of the Literature", *Adaptive Behavior* vol 11, (2003) 179-203.
- [2] Marocco, D., Floreano, D., "Active vision and feature selection in evolutionary behavioural systems", *From Animals to Animats: Proceedings SAB'02*, (2002), 247-255.
- [3] Watson, J.B. "Behavior-Based Control for Autonomous Robotics". *Proc. of the 3rd Annual Conf. on Evolutionary Programming*, World Scientific, Singapore, (1994), 185-190.
- [4] Nordin, P., Banzhaf, W., and Brameier, M. "Evolution of a World Model for a Miniature Robot Using Genetic Programming". *Robotics and Auton. Systems*, V. 25, (1998), 105-116.
- [5] J. Weng, "Developmental Robotics: Theory and Experiments", *Int. Journal of Humanoid Robotics*, vol. 1, no. 2, 199-236, (2004)
- [6] J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur and E. Thelen, "Autonomous Mental Development by Robots and Animals", *Science*, vol. 291, no. 5504, pp. 599 - 600, (2000)
- [7] Asada, M., MacDorman, K. F., Ishiguro, H., Juniyoshi, Y., "Cognitive Developmental Robotics as a New Paradigm for the Design of Humanoid Robots", *Rob. and Auton. Syst.*, V. 37, pp. 185-193 (2001)
- [8] Richard A. Watson, Sevan G. Ficici, Jordan B. Pollack "Embodied evolution: Distributing an evolutionary algorithm in a population of robots". *Robotics and Autonomous Systems*, 39(1):1-18, April 2002.
- [9] Floreano, D., Nolfi, S., Mondada, F. "Co-evolution and ontogenetic change in competing robots. Advances in the Evolutionary Synthesis of Intelligent Agents", MIT Press (2001).
- [10] Ostergaard, E. & Lund, H. "Co-evolving complex robot behavior". In *ICES'03, The 5th Int. Conf. on Evolvable Systems: From Biology to Hardware*. Springer, (2003).
- [11] Duro, R. J., Santos, J., Bellas, F., Lamas, A. "On Line Darwinist Cognitive Mechanism for an Artificial Organism", *Proceedings supplement book SAB2000*, (2000), 215-224.
- [12] Changeux, J., Courrege, P., Danchin, A. "A Theory of the Epigenesis of Neural Networks by Selective Stabilization of Synapses", *Proc. Nat. Acad. Sci. USA* 70, (1973), 2974-2978
- [13] Conrad, M. "Evolutionary Learning Circuits. " *Journal of Theoretical Biology*, 46, (1974).
- [14] Edelman, G. "Neural Darwinism. The Theory of Neuronal Group Selection". Basic Books (1987), 167-188.
- [15] Bellas, F., Becerra, J. A., Duro, R.J. "Construction of a Memory Management System in an On-line Learning Mechanism", *ESANN 2006 Proceedings book* (2006), 95-100.
- [16] Toby H.J. Collett, Bruce A. MacDonald, and Brian P. Gerkey, "Player 2.0: Toward a Practical Robot Programming Framework". In *Proc. ACRA 2005*, Sydney, Australia, December 2005.
- [17] P. Caamano, R. Tedin, J.A. Becerra, "Java Evolutionary Algorithm Framework", <http://www.gii.udc.es/jeaf>
- [18] Bellas, F., Duro, R.J. "Multilevel Darwinist Brain in Robots: Initial Implementation", *ICINCO2004 Proc. Book* (vol. 2), (2004), 25-32.
- [19] F. Bellas, A. Faiña, A. Prieto, and R.J. Duro, "Adaptive Learning Application of the MDB Evolutionary Cognitive Architecture in Physical Agents", *LNAI 4095*, 434-445, 2006

An Evolving Framework for Clustering Computer Users

Jose A. Iglesias, Agapito Ledezma and Araceli Sanchis

Abstract—The idea of clustering computer users is very beneficial for making recommendations to a user based on the histories of other users with similar preferences, detecting changes in the behavior of a user, and so on. However, computer users have different needs as they learn to use a software system or their goals changes. Although there are several approaches for clustering users, most of them do not consider the changes in their behavior. In this paper, we present an approach for clustering automatically the behavior profile of a computer user and an evolving method based on *Evolving Systems* to keep up to date the created profile clusters.

I. INTRODUCTION

The idea of clustering computer users is very beneficial for making recommendations to a user based on the histories of other users with similar preferences, detecting changes in the behavior of a user, and so on. In addition, experience has shown that users themselves do not know how to articulate what they do, especially if they are very familiar with the tasks they perform. Computer users usually leave out activities that they do not even notice they are doing. Thus, only if we can observe users we can model his/her behavior correctly [1]. However, the construction of an effective computer user profile is normally a difficult problem because a human behavior is usually erratic, and sometimes humans behave differently because of a change in their goals. The latter problem makes necessary that the user profile clusters we create change and evolve.

Computer users have different needs as they learn to use a software system or their goals changes. Moreover, users needs change as they use a system and become more familiar with the task domain and its capabilities. Recent growing interest in modeling user behavior has generated an increase in research efforts on computer systems which can automatically alter aspects of their functionality to suit the needs of individuals or groups of users. However, these systems are still difficult to develop, and there are many challenges to be overcome. In this paper, we face one of the challenges of the user clustering: the creation of user clusters which can be updated dynamically.

Recent researches on user modeling have predominantly pursued users behavioral patterns or preferences, rather than on the cognitive processes that underlie that behavior. This research is also focused on this aspect in order to cluster a user behavior. We will use the approach presented in [2] for automatically creating the profile of a user based on the analysis of the sequence of commands s/he typed.

This work is partially supported by the Spanish Government under project TRA2007-67374-C02-02

Jose A. Iglesias, Agapito Ledezma and Araceli Sanchis are with the CAOS Group, Carlos III University of Madrid, Spain. E-mails: {jiglesia, ledezma, masm}@inf.uc3m.es.

Although there are several approaches for clustering users, most of the user profile groups do not change according to the environment and new goals of the users. In this research, it is proposed an adaptive approach for creating behavior profiles in order to cluster these users.

This paper is organized as follows: Section 2 provides a brief overview of the background and related work relevant to this research. Section 3 describes the construction of the user behavior profile. The evolving clustering is explained in Section 4. Finally, section 5 contains future work and concluding remarks.

II. BACKGROUND AND RELATED WORK

Different methods have been used to find out relevant information under the computer user behavior in different computer areas:

Discovery of navigation patterns: Spiliopoulou et al. [3] present the *Web Utilization Miner WUM*, a mining system for discovering navigation patterns in web sites.

Web recommender systems: Macedo et al. [4] propose a system (*WebMemex*) that provides recommended information based on the captured history of navigation from a list of known users. *WebMemex* captures information such as IP addresses, user Ids and URL accessed for future analysis.

Web page filtering: Godoy and Amandi [5] present a technique to generate readable user profiles that accurately capture interests by observing their behavior on the Web. The proposed technique is built on the *Web Document Conceptual Clustering* algorithm, with which profiles without an a priori knowledge of user interest categories can be acquired.

Computer security: Pepyne et al. [6] describe a method using queuing theory and logistic regression modeling methods for profiling computer users based on simple temporal aspects of their behavior.

In this paper, as sequences are very relevant in human skill learning and reasoning [7], the problem of user profile clustering is examined as a problem of sequence clustering. According to this aspect, Horman and Kaminka [8] present a learner with unlabeled sequential data that discover meaningful patterns of sequential behavior from example streams.

III. CREATING A COMPUTER USER PROFILE

In this research, as it was explained in [2], we consider that the commands typed by a user are usually influenced by past experiences. This aspect motivates the idea of automated sequence learning for behavior clustering; if we do not know the features that influence the behavior of an agent, we can consider a sequence of past actions to incorporate some of the historical context of the agent. Indeed, sequence learning is arguable the most common form of human and animal

learning. Sequences are absolutely relevant in human skill learning and in high-level problem solving and reasoning [9]. Taking this aspect into account in this paper, the computer user modeling is transformed into a sequence analysis problem where a sequence of commands represents a specific behavior. This transformation can be done because it is clear that any behavior has a sequential aspect, as actions are performed in a sequence.

The commands typed by a computer user are inherently sequential, and this sequentiality is considered in the modeling process (when a user types a command, it usually depends on the previous typed commands and it is related to the following commands). According to this aspect, in order to get the most representative set of subsequences from a sequence, we propose the use of a *trie* data structure [10], [11].

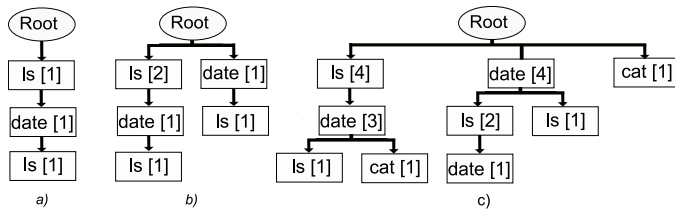


Fig. 1. Steps of creating an example trie.

The construction of a user profile from a single sequence of commands consists of three steps: 1. Segmentation of the sequence of commands, 2. Storage of the subsequences in a *trie*, and 3. Creation of the user profile. These steps are detailed in the following 3 subsections.

These steps are detailed in the following 3 subsections. For the sake of simplicity, let us consider the following sequence as an example: $\{ls \rightarrow date \rightarrow ls \rightarrow date \rightarrow cat\}$.

A. Segmentation of the sequence of commands

First, the sequence is segmented into subsequences of equal length from the first to the last element. Thus, the sequence $A=A_1A_2...A_n$ (where n is the number of commands of the sequence) will be segmented in the subsequences described by $A_i...A_{i+length} \forall i, i=[1, n-length+1]$, where *length* is the size of the subsequences created and this value determines how many commands are considered as dependent. In the remainder of the paper, we will use the term *subsequence length* to denote the value of this length. In addition, in this case, the value of this term represents how many commands a user usually types consecutively as part of his/her behavior pattern.

In the proposed sample sequence ($\{ls \rightarrow date \rightarrow ls \rightarrow date \rightarrow cat\}$), let 3 be the subsequence length, then we obtain:

$\{ls \rightarrow date \rightarrow ls\}, \{date \rightarrow ls \rightarrow date\}, \{ls \rightarrow date \rightarrow cat\}$

B. Storage of the subsequences in a trie

The subsequences of commands are stored in a *trie* in a way that all possible subsequences are accessible and explicitly represented. A node of a *trie* represents a command, 55

and its *children* represent the commands that follow it. Also, each node keeps track of the number of times a command has been inserted into it. When a new subsequence is inserted into a *trie*, the existing nodes are modified and/or new nodes are created. Moreover, as the dependencies of the commands are relevant in the user profile, the subsequence suffixes are also inserted.

Considering the previous example, the first subsequence ($\{ls \rightarrow date \rightarrow ls\}$) is added as the first branch of the empty *trie* (Figure 1 a). Each node is labeled with the number 1 which indicates that the command has been inserted in the node once (in Figure 1, this number is enclosed in square brackets). Then, the suffixes of the subsequence ($\{date \rightarrow ls\}$ and $\{ls\}$) are also inserted (Figure 1 b). Finally, after inserting the three subsequences and its corresponding suffixes, the completed *trie* is obtained (Figure 1 c).

C. Creation of the user profile

Once the *trie* is created, the subsequences that characterize the user profile and its relevance are calculated by traversing the *trie*. For this purpose, frequency-based methods are used. In particular, to evaluate the relevance of a subsequence, its relative frequency or support [12] is calculated. In this case, the support of a subsequence is defined as the ratio of the number of times the subsequence has been inserted into the *trie* to the total number of subsequences of equal size inserted.

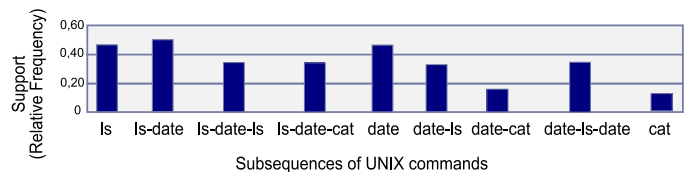


Fig. 2. Distribution of subsequences of commands - Example.

Thus, in this step, the *trie* can be transformed into a set of subsequences labeled by its support value. This set of subsequences is represented as a distribution of relevant subsequences. In the previous example, the *trie* consists of 9 nodes; therefore, the corresponding profile consists of 9 different subsequences which are labeled with its support. Figure 2 shows the distribution of these subsequences.

Once a user behavior profile has been created, it is classified in a cluster and used to update the *Evolving Clustering Library (ECLib)*, as explained in the next section.

IV. EVOLVING COMPUTER USER CLUSTERING

Clustering is the process of grouping data objects into clusters so that objects within a cluster have a high similarity in comparison to one another, but are very dissimilar to objects in other clusters. In this case, the clustering method is devised to group computer users by taking into account their distributions similarity in a feature space. The feature space is defined by distributions of subsequences of commands. Thus, a distribution in the feature space represents a specific profile which is one of the clusters (prototypes) of the evolving

clustering library *ECLib*. These prototypes are not fixed and evolve taking into account the new information collected on-line from the data stream - this is what makes the clustering method *Evolving*. The number of these prototypes is not prefixed but it depends on the homogeneity of the observed sequences. This clustering method is based in the evolving classifier proposed in [13].

The following subsections describes how a user profile is represented by the proposed clustering methods, and how the proposed clustering method work.

A. User behavior representation

First, the sequence of commands is converted into the corresponding distribution of subsequences on-line. In order to cluster a user profile, these distributions must be represented in a data space. For this reason, each distribution is considered as a data vector that defines a *point* that can be represented in the data space.

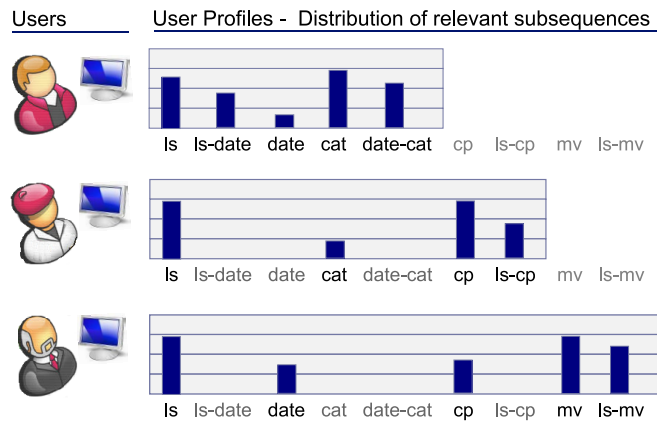


Fig. 3. Distributions of subsequences of commands in an evolving system approach - Example

The data space in which these *points* can be represented should consist of n dimensions, where n is the number of the different subsequences observed. It means that we should know all the different subsequences of the environment *a priori*. However, this value is unknown and the creation of this data space from the beginning is not efficient. For this reason, in this approach, the dimension of the data space is incrementally growing according to the different subsequences that are represented in it. Figure 3 explains graphically this idea, where the dimensions of the data space represent the different subsequences typed by the users and they will increase according to the different new subsequences obtained.

B. Structure of the clustering method

Once the corresponding distribution has been created, it is processed by the clustering method, which does not need to be configured according to the environment where it is used because it can start '*from scratch*'. Also, the relevant information of the obtained samples is necessary to update the library; but, as we will explain in the next subsection,

there is no need to store all the samples in it. The structure of this clustering method includes:

- 1) **Calculate the *potential*** of the new data sample to be a prototype (cluster).
- 2) **Update all the prototypes** considering the new data sample. It is done because the *density* of the data space surrounding certain data sample changes with the insertion of each new data sample. **Insert the new data sample** as a new prototype if needed.
- 3) **Remove** any prototype if needed.

Next 3 subsections explain each step of this evolving clustering method.

1) **Calculate the *potential* of a new data sample:** As in [14], a prototype is a data sample (a computer user profile represented by a distribution of subsequences of commands) that groups several samples which represent a certain cluster. The cluster method is initialized with the first data sample, which is stored in *ECLib*. Then, based on the *potential* of the new data sample to become a prototype, it could form a new prototype or replace an existing one.

The potential (P) of the k^{th} data sample (x_k) is calculated by the equation (1) which represents a function of the accumulated distance between a sample and all the other $k-1$ samples in the data space. The result of this function represents the *density* of the data that surrounds a certain data sample.

$$P(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} distance(x_k, x_i)}{k-1}} \quad (1)$$

where *distance* represents the distance between two samples in the data space.

In [14] the potential is calculated using the cosine distance. Cosine distance has the advantage that it tolerates different samples to have different number of attributes (in this case, an attribute is the support value of a subsequence of commands). Also, cosine distance tolerates that the value of several subsequences in a sample can be null (null is different than zero). Therefore, the proposed method uses the cosine distance (*cosDist*) to measure the similarity between two samples; as it is described in equation (2).

$$cosDist(x_k, x_p) = 1 - \frac{\sum_{j=1}^n x_{kj} x_{pj}}{\sqrt{\sum_{j=1}^n x_{kj}^2 \sum_{j=1}^n x_{pj}^2}} \quad (2)$$

where x_k and x_p represent the two samples to measure its distance and n represents the number of different attributes in both samples.

Note that the expression in the equation (1) requires all the accumulated data sample available to be calculated, which contradicts to the requirement for real-time and on-line application needed in the proposed approach. For this reason, in [14] it is developed a recursive expression cosine distance. This formula is as follows:

$$P_k(z_k) = \frac{1}{2 - \frac{1}{k-1} \frac{1}{\sqrt{\sum_{j=1}^n (z_k^j)^2}} B_k}; k = 2, 3, \dots; P_1(z_1) = 1$$

$$\text{where } B_k = \sum_{j=1}^n z_k^j b_k^j; b_k^j = b_{(k-1)}^j + \sqrt{\frac{(z_k^j)^2}{\sum_{l=1}^n (z_k^l)^2}}$$

$$\text{and } b_1^j = \sqrt{\frac{(z_1^j)^2}{\sum_{l=1}^n (z_1^l)^2}}; j = [1, n+1]$$
(3)

Using this expression, it is only necessary to calculate $(n+1)$ values where n is the number of different subsequences obtained; this value is represented by b , where $b_k^j, j = [1, n]$ represents the accumulated value for the k^{th} data sample.

2) Creating new prototypes (clusters): The proposed evolving user behavior clustering method can start 'from scratch' (without prototypes in the library) in a similar manner as *eClass* evolving fuzzy rule-based classifier developed in [14]. The potential of each new data sample is calculated *recursively* and the potential of the other prototypes is updated. Then, the potential of the new sample (z_k) is compared with the potential of the existing prototypes. A new prototype is created if its value is higher than any other existing prototype, as shown in equation (4).

$$\exists i, i = [1, NumPrototypes]: P(z_k) > P(Prot_i) \quad (4)$$

Thus, if the new data sample is not relevant, the overall structure of the clusters is not changed. Otherwise, if the new data sample has high descriptive power and generalization potential, the clusters evolve by adding a new prototype which represents a part of the observed data samples.

3) Removing existing prototypes (clusters): After adding a new prototype, we check whether any of the already existing prototypes are described *well* by the newly added prototype [14]. By *well* we mean that the value of the membership function that describes the closeness to the prototype is a Gaussian bell function chosen due to its generalization capabilities:

$$\exists i, i = [1, NumPrototypes]: \mu_i(z_k) > e^{-1} \quad (5)$$

For this reason, we calculate the membership function between a data sample and a prototype which is defined as:

$$\mu_i(z_k) = e^{-\frac{1}{2} \left[\frac{\cosDist(z_k, Prot_i)}{\sigma_i} \right]^2}, i = [1, NumPrototypes]$$
(6)

where $\cosDist(z_k, Prot_i)$ represents the cosine distance between a data sample (z_k) and the i^{th} prototype ($Prot_i$); σ_i represents the spread of the membership function, which also symbolizes the radius of the zone of influence of the prototype. This spread is determined based on the scatter [15] of the data. The equation to get the spread of the k^{th} data sample is defined as:

$$\sigma_i(k) = \sqrt{\frac{1}{k} \sum_{j=1}^k \cosDist(Prot_i, z_k)}; \sigma_i(0) = 1 \quad (7)$$

where k is the number of data samples inserted in the data space; $\cosDist(Prot_i, z_k)$ is the cosine distance between the new data sample (z_k) and the i^{th} prototype.

However, to calculate the scatter without storing all the received samples, this value can be updated (as shown in [16]) recursively by:

$$\sigma_i(k) = \sqrt{[\sigma_i(k-1)]^2 + \frac{[\cosDist^2(Prot_i, z_k) - [\sigma_i(k-1)]^2]}{k}}$$
(8)

V. CONCLUSIONS

In this paper we describe a generic approach to model and cluster automatically computer users from the sequence of commands s/he types during a period of time. As a user profile is usually not fixed but rather it changes and evolves, we have proposed a user profile cluster method able to keep up to date the created clusters based on *Evolving Systems*. This evolving cluster method is one pass, non-iterative, recursive and it has the potential to be used in an interactive mode; therefore, it is computationally very efficient and fast. Although the evaluation of this approach is not easy, we have conducted several experiments that can help us to know that we are going in a promising direction.

REFERENCES

- [1] J. T. Hackos and J. C. Redish, *User and Task Analysis for Interface Design*. Wiley, February 1998.
- [2] J. A. Iglesias, A. Ledezma, and A. Sanchis, "Creating user profiles from a command-line interface: A statistical approach," in *Conf. on User Modeling, Adaptation, and Personalization*, 2009, pp. 90–101.
- [3] M. Spiliopoulou and L. C. Faulstich, "Wum: A web utilization miner," in *In Proceedings of EDBT Workshop WebDB98*, 1998, pp. 109–115.
- [4] A. A. Macedo, K. N. Truong, J. A. Camacho-Guerrero, and M. da Graça Pimentel, "Automatically sharing web experiences through a hyperdocument recommender system," in *HYPERTEXT 2003*. New York, NY, USA: ACM, 2003, pp. 48–56.
- [5] D. Godoy and A. Amandi, "User profiling for web page filtering," *IEEE Internet Computing*, vol. 9, no. 4, pp. 56–64, 2005.
- [6] D. L. Pepyne, J. Hu, and W. Gong, "User profiling for computer security," in *Proc. American Control Conference*, 2004, pp. 982–987.
- [7] J. Anderson, *Learning and Memory: An Integrated Approach*. New York: John Wiley and Sons., 1995.
- [8] Y. Horman and G. A. Kaminka, "Removing biases in unsupervised learning of sequential patterns," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 457–480, 2007.
- [9] R. Sun, E. Merrill, and T. Peterson, "From implicit skills to explicit knowledge: a bottom-up model of skill learning," *Cognitive Science*, vol. 25, no. 2, pp. 203–244, 2001.
- [10] E. Fredkin, "Trie memory," *Comm. A.C.M.*, vol. 3, no. 9, pp. 490–499, Sept. 1960.
- [11] J. A. Iglesias, A. Ledezma, and A. Sanchis, "Sequence classification using statistical pattern recognition," in *Proceedings of Intelligence Data Analysis*. Springer, 2007, pp. 207–218.
- [12] R. Agrawal and R. Srikant, "Mining sequential patterns," in *International Conference on Data Engineering*, 1995, pp. 3–14.
- [13] J. A. Iglesias, P. Angelov, A. Ledezma, and A. Sanchis, "Modelling evolving user behaviours," in *IEEE Workshop on Evolving and Self-Developing Intelligent Systems*, 2009, 2009, pp. 16–23.
- [14] P. Angelov and X. Zhou, "Evolving fuzzy rule-based classifiers from data streams," *IEEE Transactions on Fuzzy Systems: Special issue on Evolving Fuzzy Systems*, vol. 16, no. 6, p. to appear, 2008.
- [15] P. Angelov and D. Filev, "Simpl_ets: a simplified method for learning evolving takagi-sugeno fuzzy models," *Fuzzy Systems, 2005. FUZZ '05. The 14th IEEE International Conference on*, pp. 1068–1073, 2005.
- [16] P. Angelov, X. Zhou, and F. Klawonn, "Evolving fuzzy rule-based classifiers," *Computational Intelligence in Image and Signal Processing, 2007. CIISP 2007. IEEE Symposium on*, pp. 220–225, 2007.

New Developments in Statistical Signal Processing of Quaternion Random Variables with Applications in Wind Forecasting

C. Cheong Took and D. P. Mandic¹

Abstract. Quaternions have found applications in various machine learning algorithms, however these algorithms usually do not exploit the complete available second order statistics of quaternions. To this end, we present the so-called quaternion ‘augmented’ statistics to show how to make use of the complete second order statistical information available within the quaternion domain $\mathbb{H}$. Next, as a case study, the widely linear model, which operates on the quaternion augmented statistics, is employed to enhance the performance of the Quaternion Least Mean Square algorithm. Simulations on time series prediction of both chaotic and non-stationary real world data support the approach.

1 Introduction

Recently, the resurgence in quaternion signal processing has arisen due to the advantages of quaternion algebra over real-valued quadri-variate vector algebra in the modelling of such data. In the machine learning community, quaternions have been employed in robotics, neural networks for chaotic time series prediction [2], Kalman filtering for aeronautic applications [6] and the least-mean-square estimation for the adaptive filtering of hypercomplex processes [5].

As quaternions are considered as the four-dimensional extension of complex numbers, we investigate whether the recent developments in so called augmented complex statistics can be extended to the quaternion domain, in order to cater for the generality of random signals. Pioneering works in augmented complex statistics have been undertaken by Neeser and Massey, who formulated the concept of properness (rotation invariant probability distribution) into complex-valued statistics. These researchers showed that the covariance $E\{\mathbf{z}\mathbf{z}^H\}$ of a complex random vector $\mathbf{z}$ alone is not sufficient to have a complete second order statistical description [9] and thus the pseudocovariance $E\{\mathbf{z}\mathbf{z}^T\}$ should also be considered to cater for improper signals. These theoretical findings have given way to a panoply of novel algorithms in adaptive signal processing, autoregressive moving average (ARMA) modelling, and independent component analysis [8].

Existing techniques typically consider only the information contained in quaternion-valued covariance [2, 6] and therefore, they are bound not to maximise the available statistical information. This is because quaternion-valued statistics are still their infancy [1, 11]. To shed light on quaternion statistics, Vakhania extended the concept

of ‘properness’ to the quaternion domain, however, his definition of $\mathbb{Q}$ -properness was limited to the invariance of the probability density function (pdf) under rotations of around angle of $\pi/2$ [11]. Amblard and Le Bihan improved the conditions of $\mathbb{Q}$ -properness to an arbitrary axis and angle of rotation φ , that is $[1] \ q \triangleq e^{\nu\varphi} q \quad \forall \varphi$, for any pure unit quaternion ν (whose real part vanishes); symbol $\triangleq$ means equality in terms of pdf. These results provide an initial insight into $\mathbb{Q}$ -properness, however they are applicable to only single quaternion variables and furthermore, it is not straightforward to apply them to quaternion-valued processes.

To that end, this work presents a generic framework for second order statistical analysis of the generality of quaternion-valued random variables and vectors, both second order circular and noncircular [4]. It is shown that in order to take advantage of the complete second order information, it is crucial to consider complementary covariance matrices, thus accounting for a possible improperness of quaternion processes. The advantages of the technique presented herein are likely to be similar to the benefits of the augmented statistics for noncircular complex-valued data [8]. Our analysis illustrates that the basis for augmented quaternion statistics should include quaternion involutions, and that all the necessary second order statistical information are within the so introduced augmented covariance matrix. As a proof of concept of the augmented quaternion statistics, we formulate the widely linear model in $\mathbb{H}$ to incorporate such statistics and illustrate its advantages, such as enhanced performance and improved convergence and steady state properties of the learning algorithms [3]. The theoretical findings presented herein are likely to benefit the next generation evolutionary learning algorithms.

2 Properties of Quaternion Random Vectors

2.1 Quaternion Algebra

The quaternion domain can be considered as a non-commutative four-dimensional extension of complex numbers. A quaternion variable $q \in \mathbb{H}$ consists of a real part $\Re\{\cdot\}$ (denoted by subscript a) and a vector-part $\Im\{\cdot\}$ of three imaginary components (denoted by subscripts b , c , and d), and can be expressed as:

$$\begin{aligned} q &= \Re\{q\} + \imath\Im_i\{q\} + \jmath\Im_j\{q\} + \kappa\Im_k\{q\} \\ &= q_a + \imath q_b + \jmath q_c + \kappa q_d \in \mathbb{H} \end{aligned} \quad (1)$$

The quaternion conjugate is defined as

$$\begin{aligned} q^* &= \Re\{q\} - \Im\{q\} \\ &= q_a - \imath q_b - \jmath q_c - \kappa q_d \end{aligned} \quad (2)$$

¹ Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ, U.K.; E-mail: {c.cheong-took, d.mandic}@imperial.ac.uk

$$\begin{aligned} ij &= \kappa & j\kappa &= i & \kappa i &= j \\ ij\kappa &= i^2 = j^2 = \kappa^2 = -1 \end{aligned} \quad (3)$$

Another important algebraic manipulation is the quaternion product, given as

$$q_1 q_2 = \Re\{q_1 q_2\} + \Im\{q_1 q_2\} \quad (4)$$

where

$$\begin{aligned} \Re\{q_1 q_2\} &= q_{1,a} q_{2,a} + q_{1,b} q_{2,b} + q_{1,c} q_{2,c} + q_{1,d} q_{2,d} \\ \Im\{q_1 q_2\} &= q_{1,a} \Im\{q_2\} + q_{2,a} \Im\{q_1\} + \Im\{q_1\} \times \Im\{q_2\} \end{aligned}$$

where the symbol “ $\times$ ” denotes the cross product; Notice the quaternion product inherits the non-commutativity property of the cross product.

2.2 Quaternion Involutions and the Augmented Basis Vector

It is well-known that the real and imaginary part of a complex number $z = z_a + iz_b$ to be extracted as $z_a = \frac{1}{2}(z + z^*)$ and $z_b = \frac{1}{2i}(z - z^*)$. We therefore need to employ both z and z^* to describe the elements of the corresponding signal in $\mathbb{R}^2$ and they are used as a basis for the augmented complex statistics, where the ‘augmented’ basis vector is given by $[z \ z^*]^T$. On the other hand, the quaternion domain does not allow such convenient operation and the correspondence between the elements of a quaternion valued variable in $\mathbb{H}$ and those of a quadrivariate vector in $\mathbb{R}^4$ is not clear. To circumvent this issue, we can utilise the three perpendicular quaternion involutions (self-inverse mappings), given by

$$\begin{aligned} q^i &= -iqi = q_a + iq_b - jq_c - \kappa q_d \\ q^j &= -jqj = q_a - iq_b + jq_c - \kappa q_d \\ q^\kappa &= -\kappa q\kappa = q_a - iq_b - jq_c + \kappa q_d \end{aligned} \quad (5)$$

Based on (5), we can compute the four components of the quaternion variable q as [7]

$$\begin{aligned} q_a &= \frac{1}{2}(q + q^*) & q_b &= \frac{1}{2i}(q - q^{i*}) \\ q_c &= \frac{1}{2j}(q - q^{j*}) & q_d &= \frac{1}{2\kappa}(q - q^{\kappa*}) \end{aligned} \quad (6)$$

Observe that the quaternion conjugate operation $(\cdot)^*$ is also an involution, and can be expressed as

$$q^* = \frac{1}{2}(q^i + q^j + q^\kappa - q) \quad (7)$$

Prior to the introduction of quaternion ‘augmented’ statistics, we first demonstrate the duality between the augmented quaternion vector $\mathbf{q}^a \in \mathbb{H}^{4N \times 1}$ and the quadrivariate real valued vectors in $\mathbb{R}^{4N \times 1}$. To ensure that the augmented statistics in $\mathbb{H}$ are suitable for the description of both second order circular and noncircular signals, following on (see pp. 118-119 [10]), we provide a one-to-one correspondence between the components of a quaternion variable and its quadrivariate real counterpart. For convenience, we shall utilise a combination²

² Any four of $\{q, q^*, q^i, q^j, q^\kappa\}$ or their conjugates can be used with the same effect.

$$\begin{aligned} \mathbf{q}^a &= \mathbf{A} \mathbf{q}^r \\ \begin{bmatrix} q \\ q^i \\ q^j \\ q^\kappa \end{bmatrix} &= \begin{bmatrix} \mathbf{I} & i\mathbf{I} & j\mathbf{I} & \kappa\mathbf{I} \\ \mathbf{I} & i\mathbf{I} & -j\mathbf{I} & -\kappa\mathbf{I} \\ \mathbf{I} & -i\mathbf{I} & j\mathbf{I} & -\kappa\mathbf{I} \\ \mathbf{I} & -i\mathbf{I} & -j\mathbf{I} & \kappa\mathbf{I} \end{bmatrix} \begin{bmatrix} q_a \\ q_b \\ q_c \\ q_d \end{bmatrix} \end{aligned} \quad (8)$$

where $\mathbf{I} \in \mathbb{R}^{N \times N}$ is the identity matrix, and $\mathbf{q} = [q_1 \ q_2 \ \dots \ q_N]^T \in \mathbb{H}^{N \times 1}$; similarly for $\mathbf{q}^i, \mathbf{q}^j, \mathbf{q}^\kappa \in \mathbb{H}^{N \times 1}$, and $\mathbf{q}_a, \mathbf{q}_b, \mathbf{q}_c$ and $\mathbf{q}_d \in \mathbb{R}^{N \times 1}$. The invertible $4N \times 4N$ matrix $\mathbf{A}$ provides a mapping between the augmented quaternion valued signal $\mathbf{q}^a \in \mathbb{H}^{4N \times 1}$ and the quadrivariate ‘composite’ real valued vector $\mathbf{q}^r = [\mathbf{q}_a^T \ \mathbf{q}_b^T \ \mathbf{q}_c^T \ \mathbf{q}_d^T]^T \in \mathbb{R}^{4N \times 1}$, and its inverse is given by

$$\mathbf{A}^{-1} = \frac{1}{4} \mathbf{A}^H \quad (9)$$

and in the sequel, $\mathbf{q}^r = \frac{1}{4} \mathbf{A}^H \mathbf{q}^a$. The determinant of $\mathbf{A}$ can be readily calculated as a product of its singular values, and so e.g. for $N = 1$, $\det(\mathbf{A}) = 16$. For any arbitrary N , the determinant of $\mathbf{A}$ thus becomes

$$\det(\mathbf{A}) = 16^N \quad (10)$$

The basis $\{q, q^i, q^j, q^\kappa\}$ in (8) has been chosen to make the matrix $\mathbf{A}$ unitary, thus facilitating its algebraic manipulation.

3 Quaternion Statistics

3.1 Preliminaries

Given that the mapping between quadrivariate real-valued vectors and their quaternion counterparts has now been established, we proceed on presenting the quaternion augmented statistics. In the simplest case, the elementary covariance matrix $\mathbf{C}_{\mathbf{q}\mathbf{q}}$ of a quaternion random vector $\mathbf{q} = [q_1 \ \dots \ q_N]^T$ is given by

$$\mathbf{C}_{\mathbf{q}\mathbf{q}} = E\{\mathbf{q}\mathbf{q}^H\} \quad (11)$$

The real and imaginary parts of $\mathbf{C}_{\mathbf{q}\mathbf{q}}$ are linear functions of the real-valued covariance and cross-covariance matrices of the component vectors $\mathbf{q}_a, \mathbf{q}_b, \mathbf{q}_c$ and $\mathbf{q}_d \in \mathbb{R}^{N \times 1}$.

Based on (6) and (8), the real-valued componentwise correlation matrices of the components $\mathbf{q}_a, \mathbf{q}_b, \mathbf{q}_c$ and $\mathbf{q}_d$ cannot be calculated from the quaternion-valued covariance matrix $\mathbf{C}_{\mathbf{q}\mathbf{q}}$ alone. Hence, second order information within the quaternion-valued vector $\mathbf{q}$ cannot be described fully by the covariance matrix, and we thus have to introduce complementary correlation matrices: the i -covariance $\mathbf{C}_{\mathbf{q}^i}$, the j -covariance $\mathbf{C}_{\mathbf{q}^j}$, and the κ -covariance $\mathbf{C}_{\mathbf{q}^\kappa}$. They complement the information within the covariance, and are given by

$$\begin{aligned} \mathbf{C}_{\mathbf{q}^i} &= E\{\mathbf{q}\mathbf{q}^{iH}\} \\ \mathbf{C}_{\mathbf{q}^j} &= E\{\mathbf{q}\mathbf{q}^{jH}\} \\ \mathbf{C}_{\mathbf{q}^\kappa} &= E\{\mathbf{q}\mathbf{q}^{\kappa H}\} \end{aligned} \quad (12)$$

All the components of the i -covariance $\mathbf{C}_{\mathbf{q}^i}$ are symmetric, except for the i -component $\Im_i\{\mathbf{C}_{\mathbf{q}^i}\}$ which has a skew-symmetric structure, giving rise to its i -Hermitian property. Similarly, the j -covariance $\mathbf{C}_{\mathbf{q}^j}$ and the κ -covariance $\mathbf{C}_{\mathbf{q}^\kappa}$ are respectively j -Hermitian and κ -Hermitian, meaning that

$$\begin{aligned} \mathbf{C}_{\mathbf{q}^i} &= \mathbf{C}_{\mathbf{q}^i}^H \\ \mathbf{C}_{\mathbf{q}^j} &= \mathbf{C}_{\mathbf{q}^j}^H \\ \mathbf{C}_{\mathbf{q}^\kappa} &= \mathbf{C}_{\mathbf{q}^\kappa}^H \end{aligned} \quad (13)$$

3.2 Duality between quaternionic and quadrivariate statistics

In order to recover the real-valued covariance matrices of the real part $\mathbf{z}_a$ and the imaginary part $\mathbf{z}_b$ from the complex domain, both the complex-valued covariance $\mathbf{C}_z = E\{\mathbf{z}\mathbf{z}^H\}$ and the pseudocovariance $\mathbf{P}_z = E\{\mathbf{z}\mathbf{z}^T\}$ should be employed as follows

$$\begin{aligned} \mathbf{C}_{z_a} &= \frac{1}{2} \Re\{\mathbf{C}_z + \mathbf{P}_z\} & \mathbf{C}_{z_a z_b} &= \frac{1}{2} \Im\{\mathbf{P}_z - \mathbf{C}_z\} \\ \mathbf{C}_{z_b} &= \frac{1}{2} \Re\{\mathbf{C}_z - \mathbf{P}_z\} & \mathbf{C}_{z_b z_a} &= \mathbf{C}_{z_a z_b}^T \end{aligned}$$

Similarly, we can capture the complete second order statistical description in $\mathbb{H}$, provided that the quadrivariate real-valued correlation matrices of each single component $\mathbf{q}_a, \mathbf{q}_b, \mathbf{q}_c$ and $\mathbf{q}_d$ of the quaternion random vector $\mathbf{q}$ are functions of the quaternion-valued covariance and the complementary covariance matrices as

$$\begin{aligned} \mathbf{C}_{q_a} &= \frac{1}{4} \Re\{\mathbf{C}_{q\mathbf{q}} + \mathbf{C}_{q^i} + \mathbf{C}_{q^j} + \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_b} &= \frac{1}{4} \Re\{\mathbf{C}_{q\mathbf{q}} + \mathbf{C}_{q^i} - \mathbf{C}_{q^j} - \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_c} &= \frac{1}{4} \Re\{\mathbf{C}_{q\mathbf{q}} - \mathbf{C}_{q^i} + \mathbf{C}_{q^j} - \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_d} &= \frac{1}{4} \Re\{\mathbf{C}_{q\mathbf{q}} - \mathbf{C}_{q^i} - \mathbf{C}_{q^j} + \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_b q_a} &= \frac{1}{4} \Im\{\mathbf{C}_{q\mathbf{q}} + \mathbf{C}_{q^i} + \mathbf{C}_{q^j} + \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_c q_a} &= \frac{1}{4} \Im\{\mathbf{C}_{q\mathbf{q}} + \mathbf{C}_{q^i} + \mathbf{C}_{q^j} + \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_d q_a} &= \frac{1}{4} \Im\{\mathbf{C}_{q\mathbf{q}} + \mathbf{C}_{q^i} + \mathbf{C}_{q^j} + \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_c q_b} &= \frac{1}{4} \Im\{\mathbf{C}_{q\mathbf{q}} + \mathbf{C}_{q^i} - \mathbf{C}_{q^j} - \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_d q_b} &= -\frac{1}{4} \Im\{\mathbf{C}_{q\mathbf{q}} + \mathbf{C}_{q^i} - \mathbf{C}_{q^j} - \mathbf{C}_{q^\kappa}\} \\ \mathbf{C}_{q_d q_c} &= \frac{1}{4} \Im\{\mathbf{C}_{q\mathbf{q}} - \mathbf{C}_{q^i} + \mathbf{C}_{q^j} - \mathbf{C}_{q^\kappa}\} \end{aligned} \quad (14)$$

The augmented quaternion-valued covariance matrix of an augmented random vector $\mathbf{q}^a = [\mathbf{q}^T \mathbf{q}^{iT} \mathbf{q}^{jT} \mathbf{q}^{\kappa T}]^T$ (see also (8)), is therefore defined as

$$\mathbf{C}_{\mathbf{q}}^a = E\{\mathbf{q}^a \mathbf{q}^{aH}\} = \begin{bmatrix} \mathbf{C}_{q\mathbf{q}} & \mathbf{C}_{q^i} & \mathbf{C}_{q^j} & \mathbf{C}_{q^\kappa} \\ \mathbf{C}_{q^i}^H & \mathbf{C}_{q^i q^i} & \mathbf{C}_{q^i q^j} & \mathbf{C}_{q^i q^\kappa} \\ \mathbf{C}_{q^j}^H & \mathbf{C}_{q^j q^i} & \mathbf{C}_{q^j q^j} & \mathbf{C}_{q^j q^\kappa} \\ \mathbf{C}_{q^\kappa}^H & \mathbf{C}_{q^\kappa q^i} & \mathbf{C}_{q^\kappa q^j} & \mathbf{C}_{q^\kappa q^\kappa} \end{bmatrix} \quad (15)$$

where the submatrices in (15) are computed according to

$$\begin{aligned} \mathbf{C}_\delta &= E\{\mathbf{q}\delta^H\} & \mathbf{C}_{\alpha\beta} &= E\{\alpha\beta^H\} \\ \delta &\in \{\mathbf{q}^i, \mathbf{q}^j, \mathbf{q}^\kappa\} & \alpha, \beta &\in \{\mathbf{q}, \mathbf{q}^i, \mathbf{q}^j, \mathbf{q}^\kappa\} \end{aligned}$$

To assess whether the augmented covariance matrix in (15) provides a complete second order statistical description, we need to demonstrate that it allows a static invertible one-to-one mapping with the corresponding real valued quadrivariate covariance matrix $\mathbf{C}_R$, given by

$$\mathbf{C}_R = E\{\mathbf{q}^T \mathbf{q}^{rT}\} = \begin{bmatrix} \mathbf{C}_{q_a} & \mathbf{C}_{q_a q_b} & \mathbf{C}_{q_a q_c} & \mathbf{C}_{q_a q_d} \\ \mathbf{C}_{q_b q_a} & \mathbf{C}_{q_b} & \mathbf{C}_{q_b q_c} & \mathbf{C}_{q_b q_d} \\ \mathbf{C}_{q_c q_a} & \mathbf{C}_{q_c q_b} & \mathbf{C}_{q_c} & \mathbf{C}_{q_c q_d} \\ \mathbf{C}_{q_d q_a} & \mathbf{C}_{q_d q_b} & \mathbf{C}_{q_d q_c} & \mathbf{C}_{q_d} \end{bmatrix} \quad (16)$$

Based on the mapping between the augmented quaternion-valued vector $\mathbf{q}^a$ and the corresponding real valued 'composite' vector $\mathbf{q}^r$ in (8), and that $\mathbf{q}^r = \mathbf{A}^{-1} \mathbf{q}^a = \frac{1}{4} \mathbf{A}^H \mathbf{q}^a$, the real valued covariance matrix can be calculated as

$$\begin{aligned} \mathbf{C}_R &= \mathbf{A}^{-1} \mathbf{C}_{\mathbf{q}}^a \mathbf{A}^{-H} \\ &= \frac{1}{16} \mathbf{A}^H \mathbf{C}_{\mathbf{q}}^a \mathbf{A} \end{aligned} \quad (17)$$

where $\mathbf{A}^{-H} = (\mathbf{A}^{-1})^H$. This completes the augmented quaternion statistics, which is suitable for the description of both proper and improper quaternion random processes.

3.3 Case Study: The Quaternion Widely Linear Model

To exploit the complete second order statistics of quaternion valued signals in linear mean-squared error (MSE) estimation, we need to consider a filtering model similar to the widely linear model developed for the complex case [8]. Based on (8) and (15), it is the augmented random vector $\mathbf{x}^a = [\mathbf{x}^T \mathbf{x}^{iT} \mathbf{x}^{jT} \mathbf{x}^{\kappa T}]^T$ that contains all the required second order statistical information. Then, the quaternion widely linear model (QWL) can be constructed as

$$y = \mathbf{w}^{aH} \mathbf{x}^a = \mathbf{g}^H \mathbf{x} + \mathbf{h}^H \mathbf{x}^i + \mathbf{u}^H \mathbf{x}^j + \mathbf{v}^H \mathbf{x}^\kappa \quad (18)$$

Based on this model, it can be shown that the Quaternion LMS (QLMS) algorithm [5] becomes the widely linear Quaternion LMS (WL-QLMS), whose update is given by [3]

$$\mathbf{w}^a(n+1) = \mathbf{w}^a(n) + \mu[2\mathbf{x}^a e^*(n) - e(n)\mathbf{x}^{a*}] \quad (19)$$

We now proceed on illustrating the advantages of considering the augmented statistics in the next simulation section.

4 Simulations

Three sets of simulations were conducted in a chaotic time-series prediction setting in order to demonstrate the benefits of considering the augmented statistics within quaternion-valued algorithms. As an example, the widely linear Quaternion LMS (19) which operates based on augmented statistics was assessed against the quaternion LMS [5] to predict the one-step ahead chaotic signals and nonstationary wind signals. Benchmark dataset used were the three-dimensional (3D) Lorenz attractor and the chaotic four dimensional (4D) Saito's signal [2], and a real-world 3D wind field fused with air temperature to form a four dimensional quaternion signal. The performance index was the prediction gain $Rp = 10 \log \frac{\sigma_x^2}{\sigma_e^2}$, where σ_x^2 and σ_e^2 denote respectively the estimated variances of the input and error.

4.1 Simulation 1: Three-dimensional Chaotic Lorenz Attractor

The Lorenz attractor is a three-dimensional nonlinear system used originally to model atmospheric turbulence, but also to model lasers, dynamos, and the motion of waterwheel (For more details, see p. 95 [2]). The Lorenz attractor, shown in the left plot of Fig. 2, can be regarded as a pure quaternion, and is used as a benchmark signal to test the performance of the QLMS algorithms. Fig. 1 shows the estimates of the Quaternion LMS and its widely linear counterpart, and the prediction gains measured were 30.45 dB and 32.70 dB respectively, confirming the advantage of the augmented statistics.

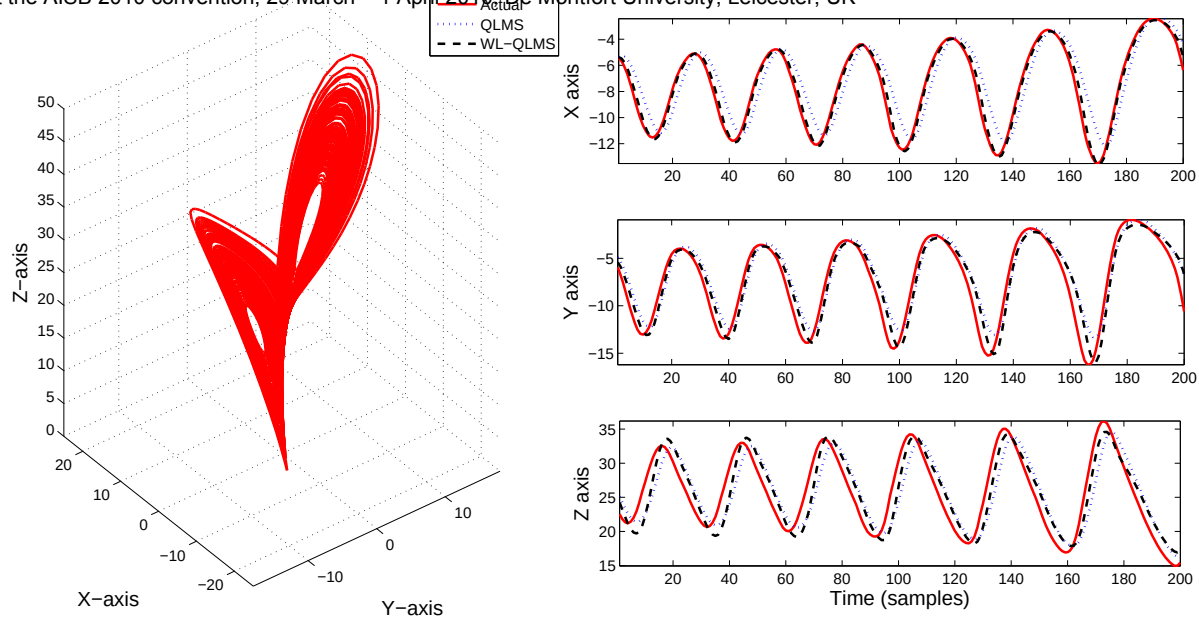


Figure 1. Prediction estimates of QLMS (dotted lines) and WL-QLMS (dashed lines) algorithms on the 3D Lorenz chaotic signal.

4.2 Simulation 2: Four-Dimensional Chaotic Saito's Process

A benchmark chaotic system (shown in left plot of Fig. 2) considered was the 4D Saito's circuit. For more details, see pp. 87-88 [2]. The right plots of Fig. 2 indicates that the widely linear Quaternion LMS ($R_p = 27.48$ dB) outperformed the Quaternion LMS ($R_p = 24.74$ dB).

4.3 Simulation 3: Wind Forecasting

To illustrate the advantage of the augmented statistics on a nonstationary real-world data, a 3D wind field (North-South, East-West and Vertical wind speed directions) was used³, together with air temperature measurements were considered in a quaternion-valued wind forecasting setting. It is clear from Fig. 3 that the augmented statistics provided enhanced performance, due to the superiority of the widely linear Quaternion LMS over Quaternion LMS. The prediction gains ($R_{p\text{WL-QLMS}} = 37.36$ dB and $R_{p\text{QLMS}} = 32.10$ dB) confirmed the observation.

5 Conclusion

Second order statistics for general quaternion-valued random variables and processes have been revisited. To make use of complete information within quaternion-valued second order statistics, complementary statistical covariances, the ν -covariance, the γ -covariance, and the κ -covariance matrices have been employed. As a case study, we have demonstrated the superiority of the WL-QLMS over QLMS on both synthetic and real world signals, thus supporting the analysis.

³ The wind data were recorded by Prof. Kazuyuki Aihara and his team at the University of Tokyo, in an urban environment. The wind data was initially sampled at 50 Hz, but re-sampled at 5 Hz for simulation purposes.

REFERENCES

- [1] P. O. Amblard and N. LeBihan, 'On properness of quaternion valued random variables', in *Proceedings of International Conference on Mathematics (IMA) in Signal Processing*, Cirencester, 23–26, (2004).
- [2] P. Arena, L. Fortuna, G. Muscato, and M. G. Xibilia, *Neural Networks in Multidimensional Domains*, Lecture Notes in Control and Information Sciences (Springer Verlag), vol. 234, 1998.
- [3] C. Cheong-Took and D. Mandic, 'A Quaternion Widely Linear Adaptive Filter', *Submitted*, (Dec. 2009).
- [4] C. Cheong-Took and D. P. Mandic, 'Second-Order Statistics of Quaternion Random Processes', *Submitted*, (Nov. 2009).
- [5] C. Cheong-Took and D. P. Mandic, 'The Quaternion LMS Algorithm for Adaptive Filtering of Hypercomplex Real World Processes', *IEEE Transactions on Signal Processing*, 57(4), 1316–1327, (2009).
- [6] D. Choukroun, I. Y. Bar-Itzhack, and Y. Oshman, 'Novel Quaternion Kalman Filter', *IEEE Transactions on Aerospace and Electronics Systems*, 42(1), 174–190, (2006).
- [7] T. A. Ell and S. J. Sangwine, 'Quaternion Involutions and Anti-Involutions', *Computers & Mathematics with Applications*, 53(1), 137–143, (2007).
- [8] D. P. Mandic and Vanessa Su Lee Goh, *Complex Valued Nonlinear Adaptive Filters: Noncircularity, Widely Linear and Neural Models*, John Wiley and Sons Ltd, 2009.
- [9] F. Neeser and J. Massey, 'Proper Complex Random Processes With Applications to Information Theory', *IEEE Transactions on Information Theory*, 39(4), 1293–1302, (1993).
- [10] B. Picinbono, *Random Signals and Systems*, Prentice Hall, 1993.
- [11] N. N. Vakhania, 'Random vectors with values in quaternion Hilbert spaces', *Theory of Probability and its Applications*, 43(1), 99–115, (1999).

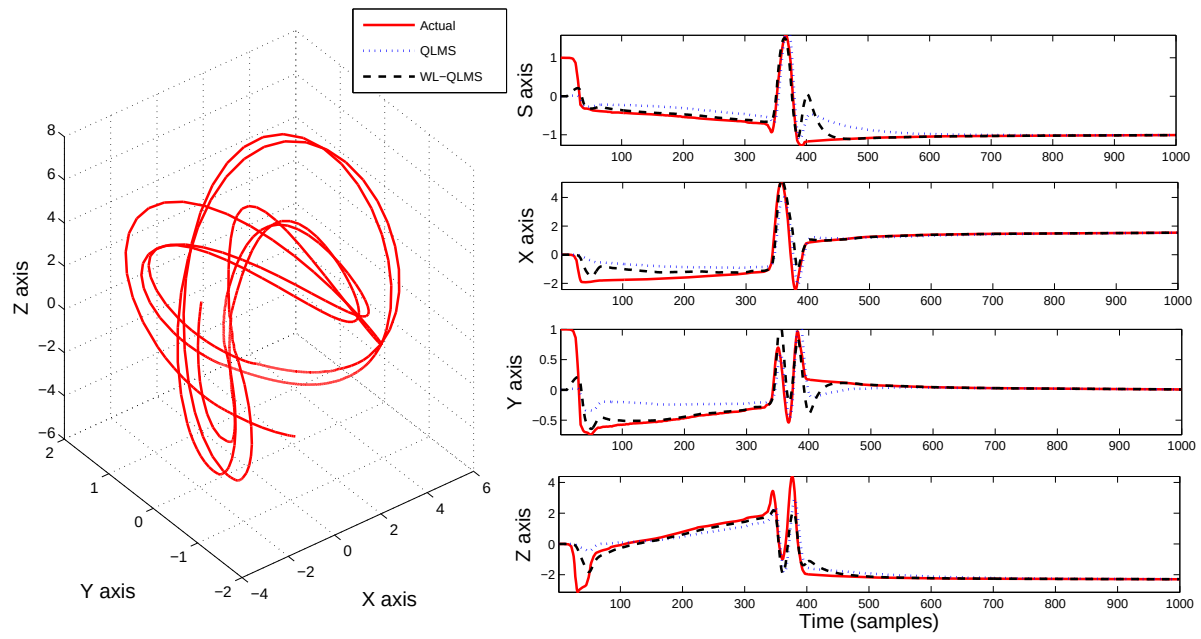


Figure 2. Prediction estimates of QLMS (dotted lines) and WL-QLMS (dashed lines) algorithms on the 4D Saito's signal.

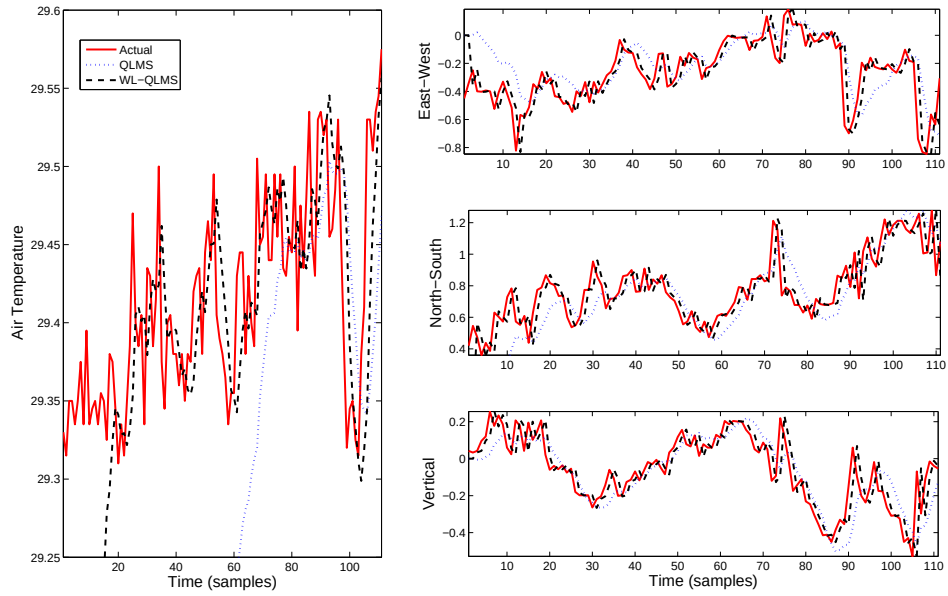


Figure 3. Prediction estimates of QLMS (dotted lines) and WL-QLMS (dashed lines) algorithms on 3D wind field fused with air temperature.

Application of ANN-GA Hybrid to run a conveyor control system

Paul Morley*

Department of Design & Innovation
The Open University
Milton Keynes, MK7 6AA, England
pm4958@tutor.open.ac.uk

Jeff Johnson

Department of Design & Innovation
The Open University
Milton Keynes, MK7 6AA, England

ABSTRACT

This paper describes the use of a genetic algorithm to find the link weights of an Artificial Neural Network used to control an industrial conveyor system, and further proposes a control strategy to allow continual adaptation of the system to changing circumstances thus maintaining the optimization of the system. Results indicate feasibility of this approach, and that performance of the ANN can be as good as conventional systematic controls

Key words

artificial neural networks, evolutionary computation, machine learning, genetic algorithm, self-adapting controls

1. INTRODUCTION

Artificial neural networks (ANNs), and genetic algorithms (GAs), are both well known and widely used in artificial intelligence fields, including their use in controls applications.

ANNs are useful for pattern recognition, and generalizing patterns in complex multi-dimensional space. GAs are useful for searching widely in large solution spaces.

The problem with ANNs, is finding the weights of the links. Various methods have been tried, the most widely known being back-propagation, which is essentially an error minimization (hill climbing) algorithm. However, there is a key limitation of this approach: it requires a set of training data which is already classified (in order to present an input pattern with a required output).

In an industrial context, conventional control systems are often set-up once achieving a reasonable level of performance (not necessarily optimal), and then left forever. However circumstances change, and in the context described here the mix of work coming through the system will drift causing a gradual fall-off in performance. Therefore it was considered that an learning system which could constantly adapt to the circumstances would be beneficial.

This paper explores the use of a hybrid system for controlling an industrial conveyor system. A single ANN is used as a black-box decision maker about the next operation to be executed, and a GA is used to find the link weights of a population of ANNs.

2. NEURAL NETWORKS

Neural Networks were first developed by McCulloch & Pitts, inspired by biological neural systems. The idea that of computing using a large number of relatively simple units working in parallel as opposed to the conventional paradigm of sequential computing is intuitively attractive.

There is also proof that in theory at least, a 2 layer network is able to implement any measurable function to an arbitrary degree of accuracy. This important result was first established by Kolmogorov (the Kolmogorov existence theorem is actually not very useful in practice, because of the unknown nature of the functions used by each unit) and then in a manner more directly relevant to ANN development by Hornik et al. [1]. Although these results show that a 2 layer (i.e. 1 hidden layer) network could in theory implement any function, in practice it may be the case that a more efficient implementation could be found by using more layers.

There are many forms of ANNs. In this experiment the a fairly simple feed-forward network was used with 3 processing layers. This network comprises layers of individual units fully interconnected to the next layer, each unit implementing a non-linear sigmoid function of a weighted sum of all the inputs

Traditionally the weights associated with the links between each neuron are found by a back-propagation algorithm, which is essentially an error minimization (hill climbing) algorithm. Whilst this is undoubtedly effective, it assumes that there is a Training Set of pre-classified data samples which can be used to train the network (set the weights) and then a separate testing set. This is known as Supervised learning. [2]

3. GENETIC ALGORITHMS

GAs provide a powerful way of exploring a complex solution space. Essentially a GA depends on being able to describe a system by a sequence of symbols - by analogy: a chromosome. Different system's chromosomes can be split and combined to create a new generation. Some form of fitness function is then used to select the 'best' individuals systems and these go forward to create the next generation and so on. A random operation is also usually introduced analogous to genetic mutation.

In essence, Genetic Algorithms tend to include the following features;

- a population of multiple instances of trial solutions to the problem in hand
- some way of determining the relative fitness or effectiveness of each solution
- some operator(s) which produce new solution instances based on the more effective previous instances.

There are lots of variations to this approach; the operators used can vary - for example, mutation & recombination are two common operators inspired by biological genetics, but are not the only ones possible. Equally, there are many different approaches for managing the population - the strongest instances in one generation may be retained in the next, or the entire population may be replaced with new instances - to name just two possibilities.

A GA will not guarantee to find the optimum solution, or even any solution. However, they tend to be effective in homing in on some effective solution, and are especially useful in extremely large search spaces. They are often called evolutionary algorithms; and although the analogy with biological evolution is strong, the actual mechanisms of biological evolution are far more subtle than these approximations.

4. HYBRID METHODS

Genetic Algorithms have been applied to both the problems of finding link weights of an ANN, and finding the an effective structure of a network with researchers reporting schemes which alternatively evolve only the structure of the network, leaving the problem of finding the weights to a deterministic approach - such as classical back-propagation: a gradient descent optimization algorithm, or keeping the structure fixed and evolving the weights. [3] provides a general review of this field.

For example, [4] describes an algorithm used to evolve both the weights and the structure of ANNs. See [5] for an earlier example which also used evolutionary programming to evolve both the weights and structure. This latter refers specifically to a system where the selection process acted on only the output of the algorithm and not on the ideas underlying the

output. That this is effective and efficient is contrary to the view of [6]. [7] and [8] give further examples of the use of a GA with selection based only on overall performance.

5. INDUSTRIAL CONTEXT

The system was applied to control a conveyor system in the context of an industrial laundry to a set of dryers. The general layout is shown in figure 1.

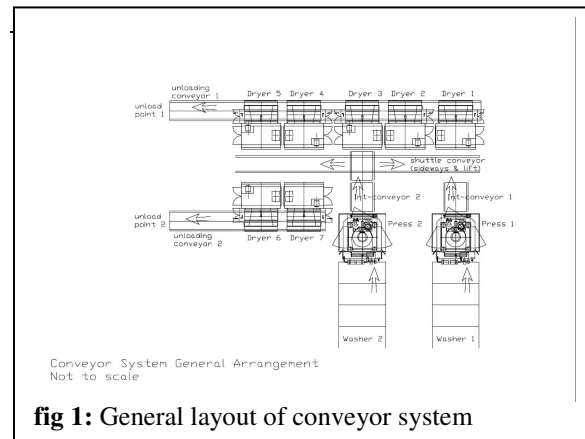


fig 1: General layout of conveyor system

System Description

This is a commercial laundry installation processing up to 6000kg of hospital linen per hour. The system was commissioned in November 2005, in a private facility in Ireland.

Batches of work move through the system, from the washers, to the water-extraction presses, one-batch storage conveyor, and into one of the dryers. The performance is affected by the shuttle conveyor which takes batches of work from one of the wash-lines, and takes them to one of the dryers. Performance is affected due to the transit time of the shuttle - for example:

- if the shuttle is in position for dryers 5 & 6, and a load of linen becomes available from washer 1, then washer 1 cannot unload until the shuttle has moved to that position thus losing that time for the washer.
- some types of work take longer in the dryers than others. If a shuttle puts long-drying work into a dryer close to the washers, then it ties up that dryer for a long time, hence dryers further away have to be used leading to longer transit times.

Generally best performance is achieved by putting longer dry-time loads in dryers further from the washers, and then parking the shuttle in front of the next washer to unload ready for it. This is the basis of conventional control algorithm. More subtle effects still cause sub-optimal performance, such as the tie-up

of the dryer unload conveyors caused by loads from - for example - dryers 1, 2 & 3 holding up the unloading of dryers 4 & 5.

Optimal utilization would achieve 30 batches from each washer per hour. Conventional control typically achieves 85-90% of this. The mix of work (with different proportions of batches with different dry times) varies over time and this causes a variation of performance.

6. HYBRID CONTROL ARCHITECTURE

A new control paradigm is proposed which has two phases:

- initial set-up
A population of ANNs is generated and a GA is used to evolve to the point where the most successful ANN can be used to run the system. This is analogous to initial commissioning of the system and can be done quite quickly simulating the system on a computer.

This paper describes work on this phase only.
- adaptive running
The controlling ANN runs the system but a population of alternative ANNs is maintained evolving with live data. When an ANN from this population consistently achieves better results than the current controlling ANN, it supplants it. In this way the control will constantly adapt to changing circumstances

Some advantages of this architecture over conventional controls are;

- conventional controls require a full analysis of the operation of the controlled system. In this case this is possible but may not always be so
- conventional controls do not adapt to changing circumstances (the initial analysis may not hold for changed circumstances)
- the method is relatively easy to implement, and also takes very little adaptation to apply to control systems in other contexts.

However the disadvantages include;

- optimal control may not be attained, and without the system analysis, the optimal performance may not be known
- the operations of the system cannot be easily explained, could not be guaranteed so would not be suitable for safety critical systems

7. APPLICATION

A population of 50 ANNs was generated, each with 2 hidden layers of 28 & 13 units. The inputs to the ANN were parameters representing the state of the system -

for example - a number representing the current load in the press, on the conveyor etc.

Each ANN was represented as a sequence of its link weights, and the GA was applied to this sequence on the following algorithm;

- run each ANN in turn for a simulated 2 hours operation and measure performance (loads produced)
- always keep best performing ANN
- all ANNs with less than average performance, eliminate from the population
- replace eliminated ANNs with new one generated by single point crossover from two parent ANNs each with better than average performance. Also apply a small mutation factor.

Fitness Function

To determine the performance of the ANN, a simple measure of how many batches the system had produced in the timescale was used. This method of measuring is remote from the individual decision outputs from the ANN. The key point is that the optimum output of the ANN for each individual decision made is not known without encountering the disadvantages discussed earlier. However the overall production of the system is known and furthermore is the parameter the operator of the system really needs to maximize.

Note that if the optimum output of the ANN was known for each decision, then simple gradient based learning would be appropriate. However, the optimum output could only generally be obtained following a full analysis of the system and this carries the disadvantages mentioned earlier.

8. ISLANDS

It was decided to implement an island strategy for the GA. This follows the parallel population strategy of [10]. This was carried over several phases;

- phase 1 5 populations of 50 ANNs were evolved in isolation for 100 generations, with differing mutation rates in each population
- phase 2 2 populations of 50 ANNs were evolved in isolation for 100 generations. Each population was formed from individuals mixed from the phase 1 populations
- phase 3 1 population of 50 ANNs taken from a mix from the phase 2 populations. Evolution was run for 400 generations.
- phase 4 the phase 3 generation was run for a further 500 generations, with a mutation rate steadily declining from 0.5% down to 0.1%

9. RESULTS & DISCUSSION

Figure 2 shows the increase in performance over the series, with a measure taken at the end of each set of 100 generations. The two curves show the performance of the best individual network in the final generation of the run, and the average of all networks over the final generation.

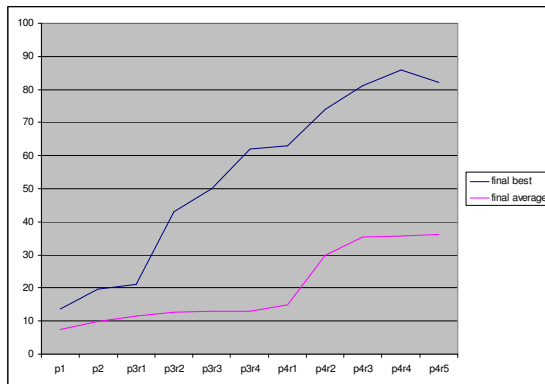


fig 2: performance (number of batches produced in the test time) of final best and final average ANNs for each evolutionary phase

The vertical axis shows the number of batches produced, over the trial period which was 5400 seconds (1.5 hours). The theoretical maximum production in this time is 90 batches. It can be seen that after the various phases of evolution, the best performing networks are able to reach performances of well over 80, which is a very respectable system utilization. (Conventional control typically achieves 85-90% of this, or 76-81 batches).

Care must be taken not to overstate these results - although the evolved controls achieve a best here of 85 batches, on occasions (due to the mix of work coming through the system) the conventional control can achieve the maximum 90 batches. This is a stochastic system. However this does indicate the credibility of evolution for generating ANN system controls, which can be comparable to conventional methods.

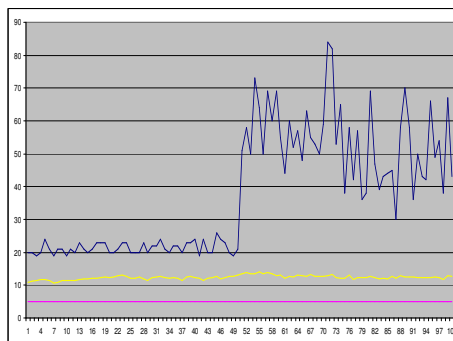


fig 3: performance (number of batches produced in the test time) of best, average, and worst ANNs.

Figure 3 shows the performance of the system during the 100 generations of phase 3 run 2. The top curve shows the best performing network in each generation. The lower curves show the average performance and the worst performance respectively. There is a step change from generation 50, (best performing network produced 21 batches), and generation 51, (best performing network produced 51 batches). Prior to this the performance had been relatively static. After this point, the best-performing network performance was relatively stable while the average performance climbed - as the successful elements of the best performing network propagated through the population. It is clear that the GA had jumped to a new and better part of the search space.

This particular result is a striking illustration of the random nature of evolutionary strategies and their ability to jump into a new area of the search space avoiding local minima.

10. REFERENCES

- [1] Hornik K, Stinchcombe M, White H, 'Multilayer Feedforward Networks are Universal Approximators', Neural Networks 2 pp359-366, 1989
- [2] Picton P, Introduction to Neural Networks, Macmillan, 1994.
- [3] Yao X, 'Evolving Artificial Neural Networks', Proc. IEEE. Vol. 87 no. 9. 1999
- [4] Curran D, O'Riordan C, 'Evolving Connect-Four Playing Neural Networks Using Cultural learning'. 2004.
<http://ww2.it.nuigalway.ie/cirg/publications.html>
- [5] Fogel D, 'Using Evolutionary Programming to Create Neural Networks that are Capable of Playing Tic-Tac-Toe'. Proc. Amer. Power Conf. 1993 pp875-879
- [6] Penrose R, 'The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics'. Penguin Books. 1989
- [7] Morley P, 'training a genetic algorithm and neural network hybrid pattern classifier by population statistics', AISB conf. 2005
- [8] Morley P, 'evolving neural networks to play noughts & crosses', ISKE conf. 2009.
- [9] Wolpert D H and Macready W G, "No free lunch theorems for optimization," IEEE Trans. Evolutionary Computation, vol. 1, pp. 67-82, Apr. 1997
- [10] Whitley D, Rana S, and Heckendorn R. 'Island Model Genetic Algorithms & Linearly Separable Problems'. Journal of Computing and Information Technology. 1998.