Digital Hardware Realization of a Novel Adaptive Ink Drop Spread Operator and Its Application in Modeling and Classification and On-Chip Training

Sajad Haghzad Klidbary*[1], Saeed Bagheri Shouraki*, and Bernabe Linares-Barranco**

*Research Group for Brain Simulation and Cognitive Science, Artificial Creatures Laboratory(ACL), Department of Electrical Engineering, Sharif University of Technology, Azadi Avenue, Tehran,11365-11155, Iran.
** Instituto de Microelectronica de Sevilla (IMSE-CNM), Consejo Superior de Investigaciones Cientificas (CSIC) and Universidad de Sevilla, 41092 Sevilla, Spain.

Abstract: In artificial intelligence, proposing an efficient algorithm with an appropriate hardware implementation has always been a challenge because of the well-accepted fact that AI hardware implementations should ideally be comparable to biological systems in terms of hardware area. Active Learning Method (ALM) is a fuzzy learning algorithm inspired by human brain computations. Unlike traditional algorithms, which employ complicated computations, ALM tries to model human brain computations using qualitative and behavioral descriptions of the problem. The main computational engine in ALM is the Ink Drop Spread (IDS) operator, but this operator imposes high memory requirements and computational costs, making the ALM algorithm and its hardware implementation unsuitable for some of the applications. This paper proposes an adaptive alternative method for implementing the IDS operator; a method which results in a marked reduction in the algorithm's computational complexity and in the amount of memory required and hardware. To check its validity and performance, the method was used to carry out modeling and pattern classification tasks. This paper used challenging and real-world datasets and compared with well-known algorithms (ANFIS and MLP) in software simulation and hardware implementation. Compared to traditional implementations of the ALM algorithm and other learning algorithms, the proposed FPGA implementation offers higher speed, less hardware, and improved performance, thus facilitating real-time application. Our ultimate goal in this paper was to present a hardware implementation with an on-chip training that allows it to adapt to its environment without dependency on the host system (on-chip learning).

Keywords: Soft Computing, Ink Drop Spread (IDS) Operator, Fuzzy Modeling, Pattern Classification, Field-Programmable Gate Array (FPGA) Implementation.

1. INTRODUCTION

The creation of algorithms capable of simulating the human brain's computing systems has always been an extremely attractive field of research, and numerous studies have been carried out with that goal as their objective. The algorithms proposed have been widely used in many applications for tasks like function modeling, control, prediction and data analysis. Two general approaches have been adopted in this type of research, namely ANNs and fuzzy logic. Some researchers consider the direct modeling of the neural networks in living organisms to be the most desirable option [1, 2], whereas, from the fuzzy logic point of

[1] *Corresponding author, Tel: +989112834604.*
E-mail address: sajjad_haghzad@ee.sharif.edu (Preferred: sajjad.haghzad@gmail.com)

view, human brain computations can also be modeled as a group of "IF-THEN" rules with linguistic variables [3, 4]. The stability and performance of both approaches have been thoroughly discussed in literature [5-10], [11-18].

One of the most important challenges for the algorithms available to date have been hardware implementation feasibility and high speed performance in real-time and online applications. The idea of creating an artificial brain with a hardware system similar to the human brain that can be used in intelligent robots has always attracted considerable attention. It should be mentioned that without a simple computing structure it is not possible to implement the brain's computational power and learning capabilities. Therefore, it is also highly desirable for any proposed hardware to be similar and comparable with biological systems [17, 19]. The digital hardware implementations of these algorithms have most attention and they are further classified as follows: 1) FPGA-based implementation, 2) digital signal processor (DSP)-based implementation, and 3) application specific integrated chip (ASIC)-based implementation. In these classes, FPGA is one of the most commonly employed platforms for implementing different algorithms on a single chip due to its reconfigurable, parallel architecture, distributed on-chip memory and logic, and powerful design, programming and synthesis tools [20]. There are two generally accepted approaches to FPGA implementation. In the first approach, the training phase is offline (the training data is static, which is named off-chip learning) and the weights obtained are implemented on the hardware. This kind of implementation is purpose-specific [21, 22]. On the other hand, in the second approach, the hardware has a learning capability, meaning that it can be used in time-variant (non-deterministic) and non-stationary (variable parameter) systems (solution space is dynamic and new data is added continuously, which is named on-chip learning) [23-26]. The second approach is actually to create an artificial brain, albeit at the cost of larger hardware. This area of research is still a hot topic.

Recently, state of the art algorithms that using Deep Learning techniques have shown well results and have attracted significant attention[27, 28]. In order to implement these algorithms, one of the most important problems is finding suitable hardware platforms with the appropriate usage of resources in different applications (computational bottleneck). Almost most of the proposed algorithms use sophisticated mathematical calculations that are almost applicable in software environments, but such computational intensity limits their usability due to large area/power requirement. This is called the area versus precision design tradeoff. The tradeoff is to choose an appropriate balance between precision and the size/cost of the hardware resources consumed.

Multi-Layer Perceptron (MLP-BP) and ANFIS (Adaptive Neuro-Fuzzy Inference System) are examples of algorithms that perform well in modeling and classifying and are implemented on different hardware platforms [25, 29-32]. MLP, which uses the error back propagation learning rule, is one the most popular learning algorithms used in ANNs. ANFIS, which integrates both neural networks and fuzzy logic inference systems, is one of the most popular neuro-fuzzy techniques. Fuzzy logic takes into account the imprecision and uncertainty of the system that is being modeled, while the neural network endows with adaptability. Based on comparable results as well as reasonable hardware, these algorithms are well known algorithms in soft computing compared to the state of the art algorithms such as Deep Learning.

Numerous dedicated hardware implementations, based on FPGA, have been proposed in the past two decades [17, 19] and references therein. These algorithms are complex due to the use of complex mathematics (such as gradient based equation) requiring the use of approximate relationships for hardware implementation. Their hardware implementation is often purpose-specific and most of the implemented hardware has no learning capability. Furthermore, increasing the number of neurons and hidden layers in various applications (large scale problems) consume considerably more FPGA resources.

The ALM, an effective soft computing algorithm inspired by the human brain's learning capability, was first presented by Shouraki in 1997[33] as an algorithm for modeling and controlling. It has a simple structure and, unlike counterparts such as the Sugeno-Yasukawa[34] and Takagi-Sugeno[14] algorithms, less computational complexities. Thanks to its specific learning

algorithm, its convergence is faster than that of the others[35]. Murakami and Honda demonstrated the high capability of the ALM in the field of soft computing[35], and in some applications it has been reported that its execution speed is 10 times faster than that of the Takagi-Sugeno and Sugeno-Yasukawa methods. Further advantages of ALM include its fast execution time, its noise robustness and, more importantly, the fact that it does not require a recursive learning process. The use of ALM has been reported in several successful applications in fields like control[36, 37], robotics[38, 39], modeling[40-44], soft computing and artificial intelligence[41, 45-48], image processing[49], and real-time processing[50].

The main idea underlying ALM is to approximate a multi-input single-output (MISO) system with some single-input single-output (SISO) subsystems. Each of these subsystems, known as IDS planes, represents the relationship between the output and one of the inputs. From each IDS unit extracts two informative features are named as *Narrow Path* and *Spread.* Training samples are mapped by instilling ink drops to represent the relationship between the inputs and the output as patterns formed on the IDS planes. ALM implementation requires a lot of resources, which in many applications may not be available. The goal of this work is therefore to propose a novel description of IDS planes in which computational complexity and hardware area are both reduced in FPGA implementation. The description should also be suitable for systems with a larger number of inputs, thus making it possible to eliminate the computational complexity of the IDS operation and reduce the required number of memory cells. In our proposed algorithm, we describe the IDS with two vectors, one of the vectors indicates the output-input relationship (*Narrow path*) and the other indicates the degree of Belief in the occurrence of output associated with each input. The following points illustrate the algorithm's effectiveness and suitability:

1) In the original algorithm, the memory space required to implement the 2-D IDS plane is quite large and memory was not used efficiently. However, the proposed method reduces both the memory requirement (reduced from $L_x * L_y \; to \; 2 * L_x$, where L_x and L_y are the quantization levels) and the algorithm's computational complexity thanks to a new description of IDS planes. The reduction in required memory and computations enable the algorithm to deal with systems with a large number of inputs.
2) Change in the paradigm of the learning process in the IDS planes results in a considerable reduction in the number of circuit elements required in the hardware implementation. This reduction results in a smaller chip and lower power consumption than in earlier implementations.
3) Compared to analog implementations, digital implementations are often simpler and more cost efficient, with higher noise robustness and shorter design times. However, larger silicon area and more power consumption are restricting. Digital circuits also enjoy higher level of scalability, reliability, stability, repeatability, and flexibility, and these features become more important in adaptive systems (time variant) where the function of the system is dependent on its parameters.
4) Our proposed algorithm has less software and hardware execution time and needs less area and resources related to MLP and ANFIS implementation and its FPGA implementation have dynamic learning capability (on-chip learning capability).

It is known that, in the human brain, computations are performed efficiently; however, currently available brain simulation algorithms are not comparably efficient. There is, therefore, a need for faster and efficient algorithms in software and hardware environment. The contribution of our work is not only to improve ALM, but also to propose an algorithm with an appropriate learning capability while taking into account the limitations of its hardware implementation (implementation of a complete fuzzy system). Furthermore, by fewer hardware resources implementation point of view, we wanted to propose a hardware which is not application dependent but versatile enough to be employed in different applications (On-chip learning: hardware with learning capability). For these reasons, the basis of work is based on the ALM, which inspired by human brain computations and need less complex computational formulas regard to other algorithms.

The rest of the paper is organized as follows. The main concepts of ALM and IDS operators are reviewed and the studies into hardware implementations of ALM in section 2. Section 3 illustrates and evaluates our proposed algorithm on real-world datasets. The hardware implementation on an FPGA platform is presented in section 4, and finally Section 5 presents some conclusions and suggests areas for future research.

2. A BRIEF OVERVIEW OF ALM

In this section, ALM is discussed in more detail. It's worth mentioning that ALM is completely different notation than active learning that is a special scenario in semi-supervised learning in which for finding most informative samples. Due to the qualitative, imprecise nature of fuzzy processing and computation systems, demand has grown for soft computing methods for simulating brain behavior. ALM, which uses fuzzy methods, is based precisely on the assumption that humans have an inaccurate attitude toward their surrounding events and, instead of memorizing numbers and digits, memorize the general behavior of the system as a series of vague, imprecise images (without using the complex mathematical and numerical iterative processes). This idea is inspired by the behavior of the brain itself, where it is assumed that the relationship between environment variables is stored as images. When faced by complex issues, human beings try to find a general knowledge about the system by simplifying concepts and finding logical relationships between them. By breaking complex problems into several simpler ones, ALM increases their understandability, and by combining and establishing links between those simpler concepts, it acts as if it were improving connections between neurons in neural networks, creating similar output. The operation of this method is shown in Fig. 1. ALM consists of two parts: the updating of the IDS planes and inference.

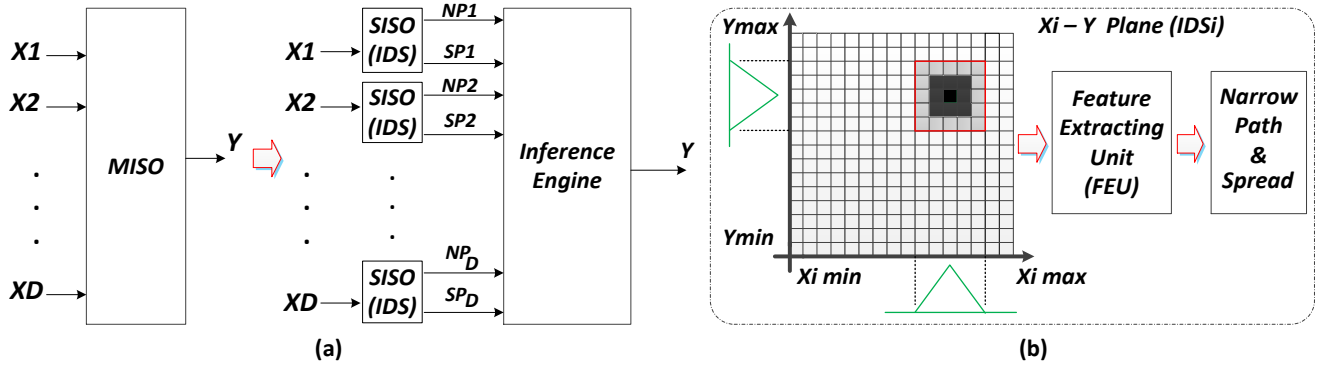

Fig. 1. (a) In ALM, the system is broken up into several SISO systems and each of these SISO systems is modeled with an IDS plane. In the inferential processing stage, all of the outputs from those systems then logically compound and the output of the whole system is obtained. (b) The effect of the IDS operator on experience points or training data as ink drops on the IDS plane. *Narrow Path* and *Spread* features are extracted from any of those planes.

2.1. IDS Operator

The core of the ALM is the IDS operator, which is implemented by means of a fuzzy interpolation and curve-fitting technique, and models uncertainty. The IDS operator works on the basis that events have a continuous nature and learning patterns are not limited to experimental data, and that their neighborhoods have information with lower degrees of uncertainty. Assuming the quantized input and output space, each of the subsystems is a gridded plane as shown in Fig. (2-a).

To see how the IDS operator works, we assume $T_1(x_1, x_2, y) = (5,8,6)$ and $T_2(x_1, x_2, y) = (9,7,10)$ as learning data in a two-input single-output system. In the first step, the ALM for the system assumes two planes, such as $(x_1 - y)$ and $(x_2 - y)$. In the second step the IDS operator drops the ink. In $(x_1 - y)$ the quantized gridded plane projects $T_{11}(x_1, y) = (5,6)$ and $T_{21}(x_1, y) = (9,10)$ and in $(x_2 - y)$ the quantized gridded plane projects $T_{21}(x_2, y) = (8,6)$ and $T_{22}(x_2, y) = (7,10)$. For simplicity, only one IDS plane is shown in Fig. (2-a).

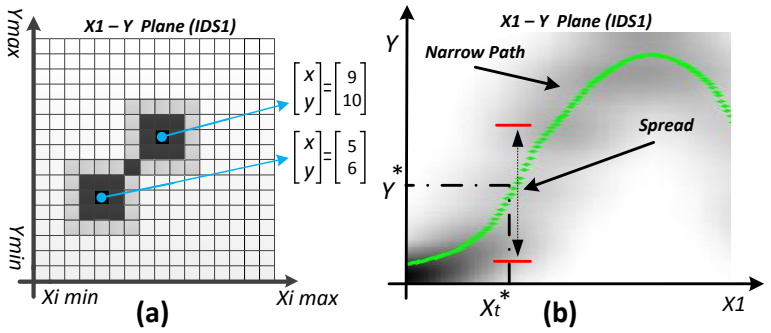

Fig. 2. IDS functionality. (a) The structure of one IDS plane is shown. When new training data enters, the plane is updated. The training data related to this plane are $T_{11}(x_1, y) = (5,6)$ and $T_{21}(x_1, y) = (9,10)$. (b) When overlapping occurs with other training data, patterns are created as shown. For a hypothetical point x_t, the *Narrow Path* and *Spread* values are expressed in graphic form.

After spreading and overlapping, the ink drops create continuous shapes in the space, as shown in Fig. (2-b). *Narrow path* and *Spread* features, which both describe the relationship between the input and the output, are extracted by the extracting unit.

Membership functions corresponding to each ink drop can be assumed as being Gaussian, pyramidal, conical, or any 3-D convex functions. In these functions, the further the distance from the center of the function, the lower the degree of membership. If the ink drops overlap, the overlap point becomes darker and the degree of belief about those points increases.

2.2. Inference Engine in ALM

Once the IDS operator has acted on 2-D planes, the *Narrow Path* and *Spread* features of the patterns generated are extracted and then used in the inferential stage of ALM. The *Narrow Path* function in IDS_i reveals the relationship between the output and the ith input. The *Spread* value represents the importance of each input variable x_i compared to other variables when determining the amount of system output. A wide spread in some input domains reveals that the output is dominantly affected by other inputs rather than x_i, because the output has varied a lot, while the ith input is almost constant. If the *Spread* is wide for all input domains on an IDS plane, fuzzy partitioning the domain of other inputs is suggested in order to make the resulting sub-domain sparser. In this case, a separate IDS plane is required for each sub-domain. By splitting the input domain, more knowledge can be extracted from the IDS planes, resulting in lower approximation error.

To calculate *Narrow Path*, operators such as max, sum, and saturating sum can be used. To find the *Spread* around each *Narrow Path* point, the point's darkness or the spread radius around it can be considered. The method proposed in [40] is discussed below. Note that because of the inaccurate nature of the algorithm and the IDS operator, there is no huge difference between the final results of the methods; the choice of method ultimately depends on the hardware and the processing speed.

Consider that we have $S = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ which includes N, training data $x_i = (x_{i1}, x_{i2}, \dots, x_{iD})^T$, $x_i \subseteq R^D$. In the first step, ALM quantizes the input and the output range of each IDS plane. Assuming that $p(x, y)$ is one point on the $x_i - y$ plane with a darkness of $d(x, y)$. The training data is updated as (x_s, y_s) as shown:

$$P_{x,y} = \{p(x, y) | x \in X_i, y \in Y\}, \quad (1)$$

$$d(x_s + u, y_s + v) = d(x_s + u, y_s + v) + h(u, v), \quad -R \le u, v \le R, \quad (2)$$

in (2), R is the ink drop radius and h is the shape of the ink drop function. Functions ψ and S are the *Narrow path* and *Spread* values, respectively, and are defined by the following equations:

$$\psi_{x_i}(x) = \left\{ b \mid \sum_{y=y_{min}}^{b} d(x,y) \approx \sum_{y=b}^{y_{max}} d(x,y) \quad , b \in Y \right\}, \tag{3}$$

$$S_{x_i}(x) = \quad \max_{y \in Y} \{y | d(x,y) > TH\} - \min_{y \in Y} \{y | d(x,y) > TH\}. \tag{4}$$

The first equation implies that the *Narrow Path* value on the IDS_i plane for any given quantized input x is b, if the sum of the grids darkness values above the (x, b) grid is approximately equal to the sum of the grids darkness values below the (x, b). The second equation implies that the Spread value on the IDS_i plane for any given quantized input x is proportional to the effective width of the formed pattern on the column of grids on the coordination of x. In (4), threshold TH is the minimum acceptable darkness of grids on the IDS plane for measuring *Spread* (for modeling purposes, $TH = 0$). In the case of N-input with m_i partitions for ith input variable, the number of IDS units corresponding to ith input which is denoted by l_i and total number of IDS units, L is as follows:

$$l_i = \prod_{i'=1,\ i' \neq i}^{N} m_{i'} \tag{5}$$

$$L = \sum_{i=1}^{N} l_i = \sum_{i=1}^{N} \prod_{i'=1,\ i' \neq i}^{N} m_{i'} \tag{6}$$

Fig. 3 shows the general ALM structure for a two-input single-output system. Note that the number of fuzzy partitions on each input domain is two. The modeling relationships and rules in Fig. 3 are as follows:

$$\begin{aligned} &R_{11}: if\ x_2\ is\ A_{21}(small) then\ y\ is\ \psi_{11}, \\ &R_{12}: if\ x_2\ is\ A_{22}(big) then\ y\ is\ \psi_{12}, \\ &R_{21}: if\ x_1\ is\ A_{11}(small) then\ y\ is\ \psi_{21}, \\ &R_{22}: if\ x_1\ is\ A_{21}(big) then\ y\ is\ \psi_{22}. \end{aligned} \tag{7}$$

At the end, the final output is the weighted sum of the above rules:

$$y\ is\ \beta_{11}\psi_{11} or\ \beta_{12}\psi_{12}\ or\ \beta_{21}\psi_{21}\ or\ \beta_{22}\psi_{22}. \tag{8}$$

The output for a system which has C number of IDS planes can therefore be calculated by the following equation:

$$\beta_i = \frac{\gamma_i \times \frac{1}{S_{x_i}(x_i)}}{\sum_{j=1}^{c} \gamma_j \times \frac{1}{S_{x_j}(x_j)}}, \tag{9}$$

$$y(x) = \sum_{i=1}^{c} \beta_i \times \psi_{x_i}(x_i). \tag{10}$$

In which ψ_{x_i} is the *Narrow Path*, S_{x_i} is the *Spread* and γ_i is our degree of belief corresponding to each IDS plane (degree of each rule). As is shown in the equations, the ALM inference is the weighted sum of the *Narrow Path* values obtained from the IDS planes. In testing phase, for white vertical column grids on the x axis, the values of $\psi_{x_i}(x)$ and $S_{x_i}(x)$ are assumed to be equal to $y_{max}/2$.

The ink radius is a significant factor affecting performance, effectiveness, convergence speed, and output error. When determining the radius, the number and density of training data in the plane should be taken into account. The radius is usually considered as a percentage of the resolution, density and distribution of data in space. The number of quantization levels for each of the input and output variables are related to the type of application, speed, and the desired accuracy. Partitioning points and radius have been obtained through trial and error. By employing heuristic methods, optimum parameters can be achieved.

Thanks to its fuzzy attitude to data, ALM has an acceptable capability in noisy systems such as function modeling and control systems[51, 52]. It also has several advantages. The order in which data is entered is not important, and, unlike ANNs, updating is done locally with no need to update globally for new data. For more information regarding ALM, see the references [35, 53] to further explore.

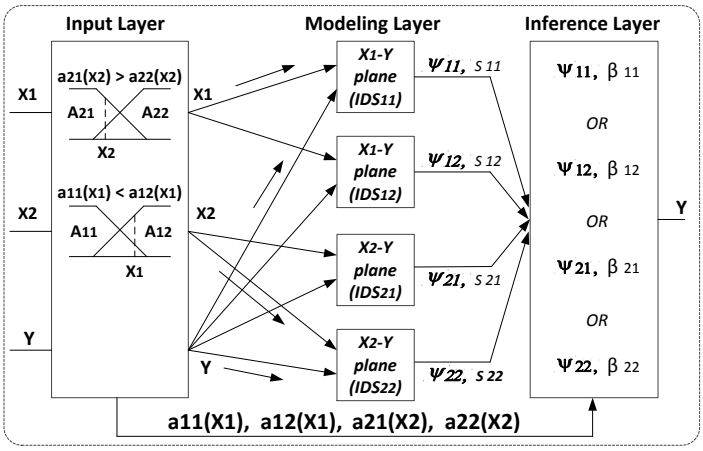

Fig. 3. General structure of ALM for a two-input single-output system. As can be seen, each of the inputs is broken up into two fuzzy partitions. After partitioning, for each partition of the first variable, one particular IDS of the second variable is considered.

2.3. RELATED WORKS ON ALM HARDWARE IMPLEMENTATION

Hardware implementation of ALM has been a challenge ever since the algorithm was first introduced. The ultimate goal is to implement an IDS chip. In this section, previous implementations of IDS are reviewed and compared.

The first hardware implementation, a quantitative structure inspired by the human retina, was proposed by Shouraki [54]. In Shouraki's proposed structure, each IDS plane was modeled by gridded planes and each grid was implemented by a photo resistor and two light emitting diodes in parallel. The advantages were that no high precision computing circuits were needed and that this structure introduced a quantitative computing paradigm. The main drawback was the structure's large hardware requirement in IDS units, which eventually resulted in poor ALM performance. To deal with this problem, Shouraki assumed two main column memory vectors X and Y, and two image column memory vectors X' and Y', in a technique called RIDS [55]. In this structure, instead of the IDS operator, the affected neighboring grids were updated by their average values. The structure hardware was based on mixed analog-digital technology, and comprised capacitive circuits, digital column memory vectors, switching circuits and ADCs. The main drawback of this method was its repetitiveness: in offline applications, for each iteration of the operator, the numbers of points were doubled (larger memory), and the averaging operation was a considerable time-consuming process in RIDS.

To deal with real-time applications, Murakami proposed High Speed IDS (HIDS) [40, 56]. In this structure, IDS planes were considered as memory cards communicating independently with the central processor through a PCI bus. Each of these cards was equipped with a controller which controlled data transfer, IDS operations and feature extraction. This structure was expensive and large, due to the employment of a PCI bus, signaling and data transfer were not performed fast enough. Actually, the structure Murakami proposed was intended only for experimental purposes in the laboratory, and not as a component of any intelligent system in the future. With the controller and the large amount of memory space it required, it was simply too inefficient.

In another study, Tarkhan proposed an analog hardware based on analog current memory circuits as a means of achieving a fast structure. By simplifying the IDS operation with a subtraction function, ink intensity was coded as current stored in an analog

memory [57]. Digital implementation is generally simpler and less expensive than analog implementation because it is less complicated and its design process, testing, simulation and synthesis are cheaper. It is also less complex [58]. The drawbacks of the aforementioned methods are, therefore, their high power consumption and the high complexity of each memory cell.

In the original ALM implementation, creating IDS planes and then storing values in the memory was time consuming, so Firouzi, adopting another approach, employed pipelined RIDS (PRIDS) [59]. In this architecture, a pipelined circuit was proposed for the averaging operation and the capacitive switching circuit was replaced by a simple digital circuit. Obviously, this structure was faster than the previous one; however, its drawback was that it used a separate division circuit for each stage of the pipeline, and this negatively affected the hardware implementation and timing parameters of the whole design. Because of iterative learning process of RIDS, it is suitable for offline applications rather than online ones.

In recent years, a memristor implementation of ALM has also been proposed. Memristor is a two-terminal element with values that vary according to the voltage applied. It acts like a memory resistor. The first memristor implementation of IDS planes was proposed by Merrikh-bayat [51] and Esmaili [60]. IDS planes require a large number of memristors, but large scale memristor-crossbar structures have not been successfully implemented. Considering the numerous advantages of digital circuits, and the problems involved in implementing large scale memristor-crossbars, the applications described in this work are limited to digital domains.

The aforementioned hardware structures all suffer from having large memory requirements for storing IDS planes, computational complexity, high power consumption and large hardware, and previous implementations of ALM have therefore also been marked by excessive hardware complexity, despite the simple nature of the algorithm. This has made their implementation almost impossible for large-scale systems. It should be remembered that without a simple computing structure it is not possible to simulate the brain's computational power and learning capabilities. Moreover, the potential of the ALM algorithm in adaptive and online applications cannot be achieved with previous hardware implementations. Therefore, we propose a novel algorithm which eliminates, to some extent, the aforementioned problems regarding precision, speed, and hardware area.

3. THE PROPOSED ALGORITHM (NAIDS/NAALM)

One fact which should be kept in mind is that the human brain has a very efficient processing; however, currently available algorithms are not comparably efficient. In this regard, the algorithm as it stands cannot be considered adequate, and faster and novel ways need to be found to solve this problem or change our attitude towards the issues that surround it. In ALM, inference is based on the features which are extracted from the IDS units. Hence, the ALM performance is directly related to the performance and implementation of the IDS units. IDS operator utilizes simple operators for instilling and overlapping ink drops, but the feature extraction unit has high computational cost. Therefore, the drawbacks of IDS computational systems are its need to huge computation, huge memory space and hardware which to store IDS planes information and extract *Narrow path* and *spread* features. A replacement therefore has to be found capable of reducing IDS computation complexities and thereby changing the learning paradigm in IDS. This would increase the algorithm's software simulation speed and improve its hardware solutions.

The first effective step toward presenting a new algorithm as an IDS replacement is to provide a suitable mapping technique capable of depicting the whole space. This mapping technique should be able to overcome the challenge of computational complexity and should utilize simple equations, capable of being implemented and used in related applications. In the proposed algorithm, the IDS planes are described by two descriptive vectors (memory vectors), where all information about the planes is stored. Based on the distances between these vectors and the training data on the IDS plane, their values shift toward the data by

Gaussian function. The two vectors are *Narrow Path (NP)* and degree of *Belief (BL)*. Like other learning algorithms, the proposed algorithm has two main phases; training and test.

Assume that the training set of S which includes N, D-dimensional training data. Each of the memory vectors of the IDS plane are shown as $v(x)$. v_{NP} is related to the output of each input that is an equivalent the variation of the *Narrow Path* in the IDS plane. v_{BL} represents the degree of *Belief* towards each of the outputs. The learning method is that when new training data is entered, the related values stored in the memory vectors are updated based on their distances with y (output) of the training data.

The initial values for v_{NP} and v_{BL} in the training phase of the algorithm are as follows:

$$v_{NP}(i) = L_y/2, \quad \mathrm{i} \in \{1, 2, 3, \dots, L_x\}, \tag{11}$$

$$v_{BL}(i) = 0, \qquad \mathrm{i} \in \{1, 2, 3, \dots, L_x\}, \tag{12}$$

in the proposed method, these vectors can have any initial values, without affecting the final convergence of output. For each input data by measuring the distance between the data and the memory vectors of each IDS plane, the vectors can be updated. Assuming that point $q(x)$ is located on the x-axis and we have(x_s, y_s) as training data, the memory vectors are updated using the following equations:

$$Q(x) = \{q(x) | x \in X_i\}, \tag{13}$$

$$v_{NP}(x_s + u) = v_{NP}(x_s + u) + \omega \times [y_s - v_{NP}(x_s)] \times g(u), -Ir \le u \le Ir, \tag{14}$$

$$v_{BL}(x_s + u) = v_{BL}(x_s + u) + \alpha \times g(u), \quad -Ir \le u \le Ir, \tag{15}$$

$g(u) = e^{\frac{(u-x_s)^2}{2\delta^2}}$ is the Gaussian function with the standard deviation δ. Ir which is called as the smoothing radius in the new IDS method(NAIDS) which has value of $3/2\delta \le Ir \le 4\delta$.

The parameters used in this memory vector are learning coefficients ω, α and δ. Learning coefficients are used to determine the effectiveness and the convergence of the IDS memory vectors towards the training data and standard deviation is used to determine how many neighbors of the training data would be affected. ω always has a value close to one and α is determined according to the distance between the input training data and v_{NP}. If this distance is less than Ir, α takes a maximum value (for example $\alpha_{high} = \omega$) and if the distance is higher than Ir, α takes a minimum value (α_{low}). The values of these parameters are directly related to the effectiveness, speed and convergence of the algorithm and are determined taking into account problems, the amount of training data and convergence speed. When the number of training data is low, ω and α should be increased and when the number of training data is high, these parameters should be decreased in order to increase output accuracy. These parameters can be calculated by trial and error methods or using optimization methods such as GA or new technique [61]. In the example shown in Fig.4, which depicts the performance and convergence of the algorithm, two hypothetical training data are driven into the system and the process of updating the memory vectors is shown.

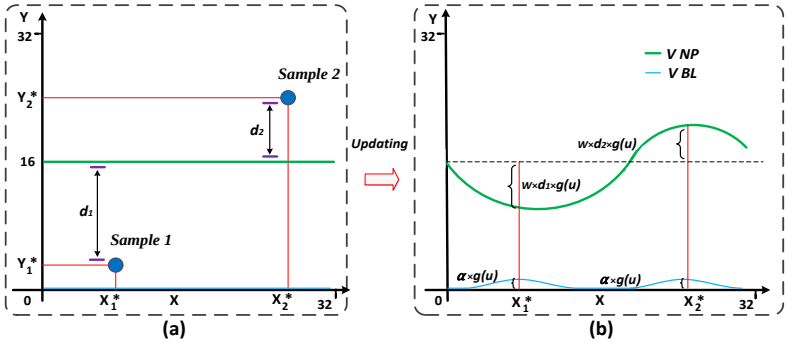

Fig. 4. The performance of the proposed algorithm for a SISO system. (a) Two quantized hypothetical training data$(x_{q1}, y_{q1}) = (7,5)$ and $(x_{q2}, y_{q2}) = (26,24)$, (quantization level is $L = 32$). (b) Vectors related to the IDS planes are updated based on the coefficient which directly correlates to the distance between the training data and the vectors. The surrounding data is updated by Gaussian function as in (14) and (15). The parameter values are $\omega = 0.4$, $\alpha = 0.9$, and $Ir = 6$.

In the proposed algorithm, data patterns in the experience space described by two separate memory vectors. Unlike in the regular IDS algorithm, which requires a memory matrix for storage, the description of the IDS space in the proposed algorithm uses a considerably smaller amount of memory (memory reduction from $L_x * L_y$ to $2 * L_x$). The IDS operator was therefore removed and its computational complexities not taken into consideration (removal of complex computations in (3) and (4)). After the learning stage, in the test stage, only the related values of the test data from the memory vectors are read and there is no need for any computation to find the features of the IDS planes. In the inference stage, the output of the system can be calculated by the following equation:

$$\beta_i = \frac{\gamma_i \times v_{BL}(x_i)}{\sum_{j=1}^{c} \gamma_j \times v_{BL}(x_j)}, \tag{16}$$

$$y(x) = \sum_{i=1}^{c} \beta_i \times v_{NP}(x_i). \tag{17}$$

As can be seen in (16), and unlike in (9), the division has been deleted and $v_{BL}(x_i)$ has the same performance as $\frac{1}{s_{x_i}(x_i)}$. Other operators, such as subtraction and division, which take up a lot of hardware volume in the desired chip, are unnecessary. It should be noted that only needs to read two memory vectors. For analyzing the computational complexity (O-notation) of algorithms, time complexity and space complexity are reported. Despite the original IDS operation that requires a $L_x * L_y$ memory matrix to store the information of each IDS plane, in the proposed approach, two vectors with length L_x are sufficient. Therefore, ALM has $S(n) = O(R_{sn}^2)$ space complexity and NAALM has $S(n) = O(R_{sn})$ space complexity. This modification results in considerable decrease in the memory requirement. ALM for each IDS unit has $T(n) = O(n \times R^2 + R_{sn}^2)$ time complexity and NAALM has $T(n) = O(n \times Ir + R_{sn})$ time complexity, where n is the number of input samples and $R(Ir)$ is the ink radius in ALM (smoothing radius in NAALM) and L is the resolution of the IDS plane. Hence, removing extraction of features and using sample-based training mode instead of batch-based training mode, results in considerable increase in the speed and decrease in the hardware. Therefore, the complexity of the NAALM is less than ALM. In the rest of the paper, an ALM algorithm with its IDS unit replaced by an NAIDS unit is referred to as an "NAALM".

3.1. Evaluation and Results

In this section, in order to analyze the proposed algorithm, some simulations were carried out to test various standard benchmarks. We only compare NAALM with ALM, ANFIS and MLP. Due to, in the next section, we want to investigate hardware implementation, we only choose the algorithms that their implementation be fair and comparable with our proposed algorithm not with the state of the art algorithms that their implementation are difficult and need a lot of resources and memories (algorithms with fewer number of neurons have been chosen for fair comparison). We first discuss modeling applications and then discuss classification. For more precision in some simulations, the genetic algorithm can be used. All of the simulations are done by MATLAB 2013 environment and Neural Network and Fuzzy Logic Toolbox with Core i5 processor, 2.4 GHz, and 4GB RAM (a personal computer).

3.2. Modeling

Function modeling is utilized in several fields, including control, estimation and complex system modeling. This section is about the system modeling of Y_1 and Y_2, two non-linear two-input, single-output systems.

$$Y_1 = (1 + x_1^{-2} + x_2^{-1.5})^2 \quad , 1 \le x_1, x_2 \le 10 \tag{18}$$

$$Y_2 = (x_1 - 6 \times \mathrm{Sin} x_2)^2 \quad , 1 \le x_1, x_2 \le 10 \tag{19}$$

To examine the algorithm's accuracy and modeling error, FVU (Fraction of Variance Unexplained) and Pearson Correlation Coefficients (PCC) were used, as in [35, 51]. The corresponding equations are as follows:

$$FVU = \frac{\sum_{i=1}^{k}(y^e(x_i) - y(x_i))^2}{\sum_{i=1}^{k}(y(x_i) - \bar{y})^2}, \qquad \bar{y} = \left(\frac{1}{k}\right)\sum_{i=1}^{k} y(x_i), \tag{20}$$

$$PCC = \frac{\sum_{i=1}^{k}(y_i - \bar{y}) \times (y_i^e - \overline{y^e})}{\sqrt{\sum_{i=1}^{k}(y_i - \bar{y})^2 \times \sum_{i=1}^{k}\left(y_i^e - \overline{y^e}\right)^2}}. \tag{21}$$

In equation (20), y^e is the output of the model, y is the output of function, k is the number of test data. In equations, $\bar{y}$ and $\overline{y^e}$ are the averages of two vectors. An accurate model has a low FVU value and a PCC close to one. Fig. 5 shows two functions Y_1 and Y_2 and the model approximated by an NAALM.

The performance of the proposed algorithm was evaluated and compared with ALM, MLP and ANFIS, and the results are shown in Table 1. In MLP network simulations, 2 neurons are considered as an input, 10 neurons for the hidden layer and 1 neuron as the system output. The neuron activation function is sigmoid and the learning algorithm is the Levenberg-Marquardt (LV) algorithm, with a learning rate 0.09. In ANFIS network simulations, the FIS has been generated using the function genfis1, the number of membership functions for each input is two and the membership functions are bell-shaped. The maximum number of epochs in MLP and ANFIS is assumed 200. The inputs normalized in the interval [-1, 1]. In simulations, in modeling dataset, 70% of which were used in the training phase.

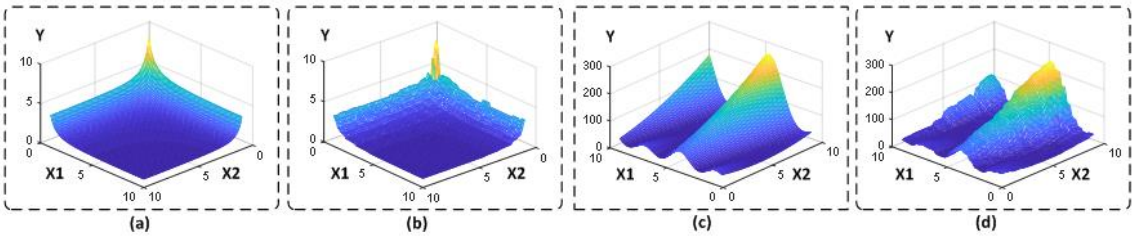

Fig. 5. (a) Function Y_1 as defined in (18). (b) The model approximated by the NAALM. The training sample size is 700. The number of fuzzy partitions in each input domain is 10. Parameter settings are as follows: $\mathrm{Ir} = 14, \omega = 0.59, \alpha = 0.28$, $L_x = L_y = 160$. In this example, FVU=0.0361 and PCC=0.9851. (c) Function Y_2 as defined in (19). (d) The model approximated by the NAALM. The training sample size is 700. The number of fuzzy partitions in each x_1 domain is 8 and in each x_2 domain, 12. Parameter settings are as follows: $\mathrm{Ir} = 14, \omega = 0.57, \alpha = 0.39, \mathrm{L_x} = \mathrm{L_y} = 200$. In this example, FVU=0.0292 and PCC=0.9879.

According to the results shown in Table 1, the proposed algorithm could therefore represents a good modeling of the functions. As reported in the table, by assuming constant learning coefficients, in order to increase the accuracy in small datasets, the radius should be increased. On the other hand, in large datasets, the radius should be decreased. Increasing the radius in large datasets creates inaccurate information from neighbors. The reason for the low accuracy of the proposed algorithm and the ALM is due to the imaging of the data on the IDS planes, which will overlook valuable information embedded in large datasets. Thanks to the absence of iterative processes in the training phase, ALM is faster than the MLP and ANFIS algorithms. NAALM is faster than ALM because the calculations associated with the feature extraction have been avoided. Hence, there must be a compromise between accuracy and speed (processing time), which, speed is a critical parameter for real-time applications. Compared to ANFIS in the modeling of non-smooth Y_2, the number of membership function should be changed for higher accuracy.

Table 1: Comparison of NAALM, ALM, ANFIS and MLP based on FVU and PCC metrics. For Y_1, in NAALM, $\alpha = 0.25$ and $\omega = 0.51$. For Y_2, in NAALM, $\alpha = 0.3$ and $\omega = 0.53$. The partitioning points in the input domain in Y_1 are 10 for x_1 and 10 for x_2, and in Y_2 are 8 for x_1 and 12 for x_2. Each algorithm is repeated 20 times and the average reported.

# Samples			**250**			**1000**		
Alg.	**Function**	**R/Ir**	**FVU**	**PCC**	**Time(s)**	**FVU**	**PCC**	**Time(s)**
NAALM	Y_1	8	0.1211 ± 0.0169	0.9392 ± 0.0125	0.0214	0.0263 ± 0.0097	0.9879 ± 0.0061	0.0708
		12	0.0831 ± 0.0154	0.9662 ± 0.0137	0.0218	0.0203 ± 0.0081	0.9912 ± 0.0052	0.0861
		16	0.0629 ± 0.0131	0.9734 ± 0.0118	0.0231	0.0211 ± 0.0091	0.9904 ± 0.0064	0.0918
	Y_2	8	0.1344 ± 0.0207	0.9331 ± 0.0189	0.0158	0.0327 ± 0.0084	0.9863 ± 0.0077	0.0721
		12	0.0951 ± 0.0180	0.9557 ± 0.0152	0.0162	0.0243 ± 0.0079	0.9872 ± 0.0066	0.0897
		16	0.0781 ± 0.0142	0.9682 ± 0.0139	0.0168	0.0268 ± 0.0086	0.9869 ± 0.0069	0.0922
ALM	Y_1	8	0.1112 ± 0.0163	0.9467 ± 0.0127	0.1780	0.0223 ± 0.0083	0.9852 ± 0.0082	0.2192
		12	0.0872 ± 0.0159	0.9645 ± 0.0134	0.1811	0.0217 ± 0.0076	0.9901 ± 0.0078	0.2304
		16	0.0713 ± 0.0144	0.9698 ± 0.0115	0.1979	0.0293 ± 0.0092	0.9864 ± 0.0085	0.2620
	Y_2	8	0.1256 ± 0.0184	0.9388 ± 0.0176	0.2258	0.0289 ± 0.0093	0.9811 ± 0.0082	0.2539
		12	0.0848 ± 0.0172	0.9656 ± 0.0147	0.2342	0.0219 ± 0.0089	0.9861 ± 0.0077	0.2807
		16	0.0745 ± 0.0133	0.9702 ± 0.0131	0.2403	0.0241 ± 0.0090	0.9842 ± 0.0080	0.2960
ANFIS	Y_1	-	0.0079 ± 0.0028	0.9964 ± 0.0019	0.3173	0.0042 ± 0.0011	0.9980 ± 0.0012	1.1808
	Y_2	-	0.1116 ± 0.0339	0.9456 ± 0.0143	0.3226	0.0875 ± 0.0082	0.9656 ± 0.0044	1.1455
MLP	Y_1	-	0.0189 ± 0.0731	0.9904 ± 0.0376	1.8985	0.0015 ± 0.0020	0.9993 ± 0.0010	3.9697
	Y_2	-	0.0129 ± 0.0345	0.9937 ± 0.0174	1.6431	0.0025 ± 0.0036	0.9988 ± 0.0018	4.6756

As mentioned before, by breaking the domain of the input data, the error will be reduced to a good extent. Moreover, learning coefficients (α and ω) have an improving effect on accuracy. In Fig. 6, some experiments on modeling dataset show how these parameters affect the modeling results. As shown in the Fig. 6, with regard to the number of training data, appropriate values can be obtained for parameters to converge faster and result in lower error rates. As shown, the curves related to different parameters have a minimum value that the best convergence happened for that value. Determining the radius, learning parameters and the number of partitions are affected by the size of the training set and the density of data points in the feature space.

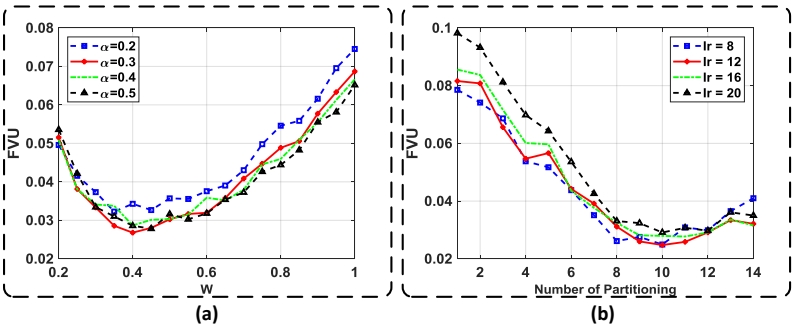

Fig 6. (a) The Comparison proposed method based on FVU with respect to different value of learning coefficients (α and ω) and $Ir = 16$ in approximating function of Y_1. (b) The Comparison proposed method based on FVU with respect to different value of radius and number of partitioning approximating function of Y_1. Parameters are as follows: $\alpha = 0.3, w = 0.55$. In both simulations the training set size is 1000.

3.3. Classification

Undoubtedly, one of the key capabilities of the human brain is its ability to classify surrounding objects based on how those objects' features relate to each other. To examine the performance of the proposed algorithm in greater depth, therefore, evaluation in the classification problems is investigated. In all simulation we used 10-fold cross validation method for model evaluation. The Iris dataset is a real famous standard classification benchmark containing 150 data samples for three species of iris flowers (each species has 50 samples). Each data sample has four features (lengths and widths of petal and sepal). Two of the three classes are not linearly separable, as can be seen in Fig. (7-a).

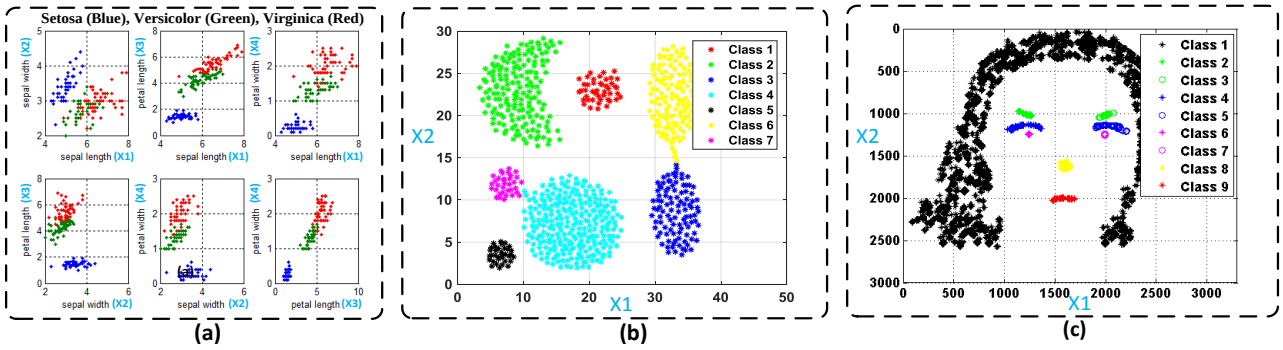

Fig. 7. (a) Iris flowers dataset, divided into three classes: Setosa (blue), Versicolar (green), and Virginica (red). Each sample has four features: Sepal length, Sepal width, Petal length and Petal width. (b) Aggregation dataset divided seven different classes and each sample has two features (788 data samples in all). (c) Girl dataset divided nine different classes and each sample has two features.

The classification accuracy of different algorithms for Iris, Aggregation, and Girl dataset are shown in Table 2. In Iris dataset, the whole system was modeled as a four-input, single-output system with four IDS planes in the first step being broken up depending on the required accuracy. In NAALM, $\alpha = 0.44$, and $\omega = 0.77$. The fuzzy partitioning in the input domains is 4 for ALM and NAALM algorithms. The quantization levels are 256 and 100 in ALM and NAALM, respectively. In MLP network simulations, 4 neurons are considered as an input, 7 neurons for the hidden layer neurons and 1 neuron as the system output. The neuron activation function is sigmoid and the learning rate is 0.03. In ANFIS network simulations, the FIS has been generated using the function genfis2. Number of epochs are 100 and 400 in MLP and ANFIS, respectively.

Table 2: Classification results of ALM, NAALM, ANFIS and MLP in the Iris, Aggregation, and Girl datasets. Each algorithm was repeated 20 times and the average reported. The processing time is to compare the speed of proposed algorithm.

Dataset	**Iris**			**Aggregation**			**Girl**		
Algorithm	**R / Ir**	**Accuracy %**	**Time (S.)**	**R / Ir**	**Accuracy %**	**Time (S.)**	**R / Ir**	**Accuracy %**	**Time (S.)**
	12	95.33 ± 5.49	**0.0319**	**16**	97.34 ± 1.09	**0.0253**	**4**	98.22 ± 0.42	**0.4101**
NAALM	**14**	**97.33** ± 4.66	0.0328	**20**	**99.50** ± 0.98	0.0261	**8**	99.47 ± 0.31	0.7132
	16	94.67 ± 5.26	0.0335	**24**	96.07 ± 1.62	0.0267	**12**	98.23 ± 0.21	0.8613
	12	94.00 ± 5.84	0.0987	**16**	96.22 ± 1.13	0.2314	**4**	98.30 ± 0.13	0.6220
ALM	**14**	95.33 ± **3.22**	0.1043	**20**	99.37 ± **0.65**	0.2651	**8**	99.43 ± **0.11**	0.8951
	16	93.33 ± 5.44	0.1175	**24**	98.24 ± 0.82	0.2993	**12**	98.01 ± 0.14	1.3736
ANFIS	-	95.33 ± 4.50	0.9332	-	99.49 ± 3.03	7.8148	-	98.15 ± 0.08	390.95
MLP	-	96.67 ± 3.51	0.7847	-	99.10 ± 3.42	2.1687	-	**99.82** ± 0.28	29.892

The Aggregation and Girl dataset are shown in Fig. (7-b) and Fig. (7-c) respectively. In Aggregation dataset, in NAALM, $\alpha = 0.17$ and $\omega = 0.73$. The fuzzy partitioning in the inputs domain were 9 for ALM and NAALM algorithms. The quantization levels are 128 and 100 in ALM and NAALM, respectively. In MLP network simulations, 2 neurons are considered as an input, 22 neurons for the hidden layer neurons and 1 neuron as the system output. The neuron activation function is sigmoid and the learning rate is 0.08. In ANFIS network simulations, the FIS has been generated using the function genfis1, the number of membership functions for each input is 3 and the membership functions are bell-shaped. Number of epochs are 100 and 600 in MLP and ANFIS, respectively. In Girl dataset, in NAALM, $\alpha = 0.16$ and $\omega = 0.55$. The fuzzy partitioning in the inputs domain were 5 for ALM and NAALM algorithms. The quantization levels are 256 and 300 in ALM and NAALM, respectively. In MLP network simulations, 2 neurons are considered as an input, 24 neurons for the hidden layer neurons and 1 neuron as the system output. The neuron activation function is sigmoid and the learning rate is 0.09. In ANFIS network simulations, the FIS has been generated using the function genfis1, the number of membership functions for each input is 3 and the membership functions are bell-shaped. Number of epochs are 120 and 150 in MLP and ANFIS, respectively.

Based on the preceding tables, the proposed algorithm not only has a better speed and lower computation cost than the other three algorithms, but it also has less variance in its output (i.e., it is more stable), because, unlike the other types of algorithms, it has no need for random initialization. It is worth mentioning that NAALM and ALM algorithm are knowledge-based method, unlike neural network methods, which are black box methods. In this regard, and again unlike neural network algorithms, this algorithm has no need to preprocess the datasets when dealing with complex problems (in these examples, normalized into standard distribution), and these capabilities constitute the advantages of the proposed method in comparison with ANNs.

For more evaluation, we used challenging and real-world datasets (11 datasets). Table 3 shows the properties of all datasets and the Table 4 shows the classification accuracy of the algorithms. The four algorithms were compared on these new datasets with different number of features, samples, and classes. 10-fold cross validation method is used for model evaluation.

Table 3: This table shows the properties of all datasets that were used in our evaluation.

Dataset	# objects	# features	# classes	Source
Banana	5300	2	2	http://sci2s.ugr.es/keel/category.php?cat=clas#sub2
2-Spiral	194	2	2	http://cs.joensuu.fi/sipu/datasets
Path-based1	300	2	3	http://cs.joensuu.fi/sipu/datasets/
3-Spiral	312	2	3	http://cs.joensuu.fi/sipu/datasets
Aggregation	788	2	7	http://cs.joensuu.fi/sipu/datasets
Girl	**93000**	**2**	**9**	http://ee.sharif.edu/~acl/Projects
Haberman	306	3	2	http://sci2s.ugr.es/keel/category.php?cat=clas#sub2
Iris	150	4	3	http://sci2s.ugr.es/keel/category.php?cat=clas#sub2
Appendicitis	106	7	2	http://sci2s.ugr.es/keel/category.php?cat=clas#sub2
Wisconsin	683	9	2	http://sci2s.ugr.es/keel/category.php?cat=clas#sub2
Wine	178	**13**	3	http://sci2s.ugr.es/keel/category.php?cat=clas#sub2

Table 4: The comparison of NAALM, ALM, ANFIS and MLP based on accuracy. Acc: Accuracy, Par: Number of partitions, Res: Quantization levels, R: Ink radius in IDS, Ir: Smoothing radius of NAIDS, α and ω learning coefficients, MFs: Number of membership function, Ep: Epochs, Top: Topology of network, Lr: Learning rate.

Alg./Dataset		*Banana*	*2-Spiral*	*Path-based1*	*3-Spiral*	*Haberman*	*Appendicitis*	*Wisconsin*	*Wine*
ALM	***Acc***	$89.88 \pm \mathbf{1.89}$	$\mathbf{100 \pm 0.0}$	$\mathbf{98.88} \pm 1.85$	$99.67 \pm \mathbf{0.37}$	74.07 ± 6.89	88.05 ± 6.85	$\mathbf{97.37} \pm 1.51$	98.32 ± 2.39
	Par	*10,10*	*13,13*	*9,9*	*10,10*	*4,3,3*	*1,1,2,1,1,1,2*	*1,1,1,1,1,4,1,1,1*	*2,1,3,3,1,1,3,1,1,1,1,1,1*
	Res	160 × 160	256 × 256	90 × 90	128 × 128	200 × 200	128 × 128	128 × 128	128 × 128
	R	*16*	*10*	*20*	*18*	*6*	*12*	*12*	*7*
NAALM	***Acc***	$\mathbf{89.92} \pm 2.87$	$\mathbf{100 \pm 0.0}$	98.56 ± 2.45	$\mathbf{99.70} \pm 0.96$	73.44 ± 6.74	88.22 ± 6.97	97.13 ± 1.21	98.86 ± 2.41
	Par	*10,10*	*13,13*	*9,9*	*10,10*	*4,3,3*	*1,1,2,1,1,1,2*	*1,1,1,1,1,4,1,1,1*	*2,1,3,3,1,1,3,1,1,1,1,1,1*
	Res	140 × 140	256 × 256	90 × 90	100 × 100	150 × 150	128 × 128	160 × 160	128 × 128
	Ir	*14*	*9*	*22*	*24*	*6*	*12*	*12*	*7*
	α.	0.19	0.47	0.42	0.36	0.31	0.45	0.32	0.39
	ω.	0.39	0.89	0.68	0.96	0.37	0.87	0.74	0.88
ANFIS	***Acc***	73.21 ± 2.06	75.29 ± 9.23	98.33 ± 2.36	99.69 ± 0.99	77.11 ± 7.12	$\mathbf{96.64 \pm 1.96}$	$89.80 \pm \mathbf{0.65}$	$\mathbf{100 \pm 0.0}$
	MFs	*genfis1/3*	*genfis2/-*	*genfis1/3*	*genfis3/-*	*genfis2/-*	*genfis2/-*	*genfis2/-*	*genfis2/-*
	Ep	*800*	*1000*	*350*	*400*	*200*	*400*	*300*	*70*
MLP	***Acc***	74.34 ± 2.73	40.94 ± 13.41	96.67 ± 2.22	97.81 ± 4.18	77.33 ± 6.44	88.36 ± 3.18	94.44 ± 1.24	98.83 ± 2.29
	Top	*2-18-2*	*2-130-1*	*2-27-3*	*2-18-3*	*3-4-2*	*7-15-2*	*9-30-2*	*13-22-3*
	Ep	*200*	*1000*	*50*	*500*	*100*	*50*	*200*	*100*
	Lr	*0.04*	*0.05*	*0.07*	*0.04*	*0.03*	*0.04*	*0.06*	*0.05*

In Tables 2 and 4, in Wine, Aggregation and 3-Spiral datasets, the accuracies of all algorithms are high because of proper distance between classes. In Banana and Iris datasets, because of close classes, partitioning the inputs in ALM and AALM does

not improve the accuracy, and hence the classification accuracy has an upper bound such as MLP and ANFIS. In 2-spiral dataset, it needs an algorithm to learn a highly non-linear separation, partitioning the input domain in NAALM and ALM improves the accuracy compared to other methods. MLP has low accuracy due to the non-linearity of the data as well as the low number of training data. In Path-based1 dataset just like 2-Spiral, it needs an algorithm to learn a non-linear separation. The Haberman dataset is imbalanced, and because the ranges of some inputs are almost the same and they are in integer form, classes are found to be close to each other. In Appendicitis dataset, as the sample are in real form, ranges of all inputs are the same, and samples of each class are positioned in the same distance to each other, IDS operator of ALM and smoothing operator of NAALM does not considerably improve the accuracy, and eventually results in close classes. In Wisconsin, ALM algorithm has the highest accuracy. All algorithms have high precision in the Girl dataset (accuracy> % 98.1). A large number of samples in this dataset encountered the lack of memory and time processing (computational cost) in many algorithms. In this dataset, due to the large number of samples, the MLP and ANFIS algorithms encountered time processing problem because of their computational nature. As reported in the Table 2, NAALM was much faster than ALM and other algorithms. Therefore, in all datasets reported, the distribution of classes is the important factor in determining the accuracy of algorithms. Therefore, the proposed algorithm shows the same or in some cases better performance as the original ALM in function modeling and classification.

Noise is one of the major challenges in the domain of soft computing. The noise resistance of the different algorithms was compared and the results are shown in Fig. (8-a). The percentage of applied noise was determined by (20):

$$x_{noisy} = x + \tilde{n}_p\,,\ \ \tilde{n}_p = \frac{x \times \tilde{n} \times p}{100}, \tag{22}$$

where p is the noise percentage, $\tilde{n}$ is a uniform random number between -1 and 1 and x is the training data point. Actually, in (22) randomly displaces training data points as a percentage of their own values.

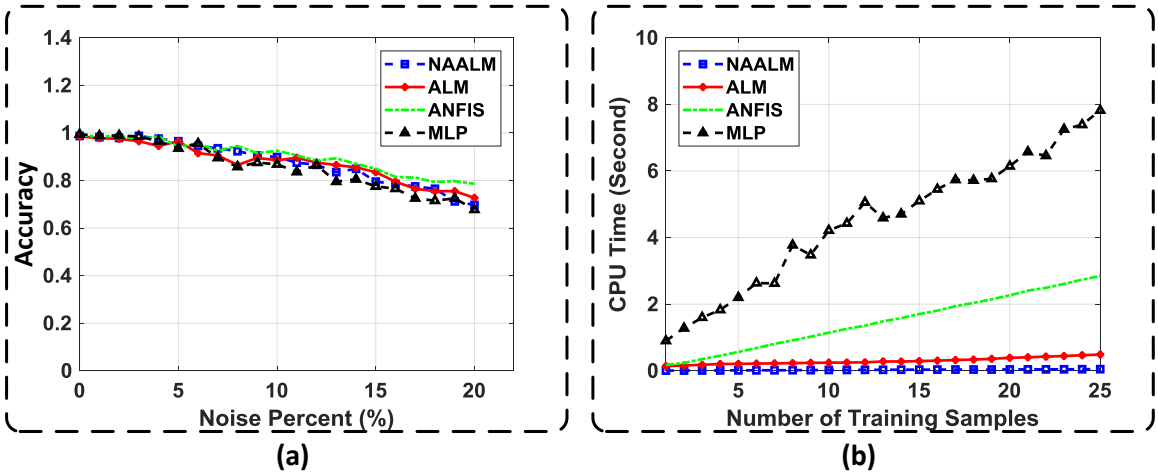

Fig. 8. (a) PCC comparison between NAALM, ALM, ANFIS, and MLP in the Aggregation dataset with respect to different percentages of applied noise. The values of the parameters are equal to the values stated in the Table 2. (b) The convergence speed of NAALM and ALM in approximating function Y_1. The values of the parameters are equal to the value stated in the Table 1.

The IDS operator and smoothing operator are comparable to the Gaussian noise that is added to the data in the training process. In this regard, the NAALM performance is similar to that of the other algorithms in terms of noise immunity. Additionally, the ALM and NAALM algorithms have less variation (more convergence stability) than ANFIS and MLP. In ANNs, depending on the initial values of weightings, different results may be obtained (producing different output weighted matrixes) and the training times might also be different. ALM and NAALM, however, are always stable and their answers converge to the almost same result. In order to compare the convergence speed of NAALM, ALM, ANFIS and MLP, function Y_1 is considered. Fig. (8-b)

compares the convergence speed with respect to the number of training samples. Fig. (8-b) shows the average processing time for 20 execution of each algorithm. NAALM is faster than ALM, because the calculations associated with the feature extraction has been avoided (its mathematical simplicity), moreover, for the test data the algorithm should read the memory vectors.

4. HARDWARE IMPLEMENTATION (EXPERIMENTAL RESULTS) ON FPGA

The building of a digital architecture in FPGA in which to implement the proposed algorithm was therefore another of this paper's objectives. In applications where online training is required, training time is often a critical parameter and speeding it up is very desirable. The main reason for realizing algorithms on FPGA is to find a solution set very fast. In this section, the hardware implementation of the proposed algorithm is presented in more detail. To implement the algorithm, the aforementioned equations in (14) and (15) had to be implemented. As is well known, multipliers are complicated modules (in terms of area, latency, and power consumption) and should be avoided as much as possible. The implementation was designed with a view to reducing multipliers in all of its steps. The implementation has been verified in several steps including MATLAB simulation, HDL based simulation, and ChipScope analyzer. The structure of the proposed hardware is shown in Fig. 9.

In Fig. (9-a), inputs x and output y are 8-bit integers. To choose the planes, two 4-bit integers called Sel_11 and Sel_12 were used to enable the IDS planes of x_1 using a priority decoder, and another two 4-bit integers called Sel_21 and Sel_22 were used to enable the IDS planes of x_2 using a priority decoder. A single (learning) bit was used to distinguish between training mode and test mode. Each IDS plane had a separate FSM (Finite State Machine) and memory controller. In practice, each data point was often mapped to more than one plane (the maximum number of active planes being four), and parallelization of the algorithm in simultaneously updating multiple planes was considered useful. The proposed hardware can work in serial mode or parallel mode (higher speed but more resources employed). In serial mode, only one plane is active, but in parallel mode up to four planes are active. The inference block was the defuzzifier subsystem which, in the test phase, uses DSP blocks to compute the weighted sum of *Narrow Path* and *Belief* values, and transfers to output (delivering a crisp output). The input control signal and the inputs are generated by using any type of programmable hardware, e.g., a microcontroller, DSP or any processor and this part is not discussed. In this implementation, the communication overhead cost between the CPU and FPGA is low and a good option is to have a System-on-Chip which includes FPGA and processors in the same chip.

Fig. (9-b) shows the FSM diagram, where controlling signals were generated to control the sequence of updating and appropriate communication with Block RAMs. This FSM had four working states: Idle (I), Begin (B), Reading (R), and Writing (W). In the Idle state, output was shown and no training was performed. In this state, the RAM block had time to call y values. When the learning signal was activated, the machine went into the Begin state. In this state, $\mathrm{y_{error}}(y_s - v_{NP})$, as mentioned in (14), was computed and stored. In the Reading state, current values of cells were read from the memory. New memory values were then computed, and when the writing signal was activated the writing process was performed.

Fig. (9-c) shows the algorithm's memory vector (descriptive vector) structure. Keeping two lines of 256-bits memory vectors (the length of memory vectors depends on assumed resolution) with Byte access requires considerable resources and its implementation is not practically feasible, especially when more than one plane is required. Two blocks of RAM were therefore used to improve the speed. This meant that one clock cycle was needed for reading and providing data. Dual-Port RAM blocks were chosen for this purpose that, in any time two cell of RAM update (the address line of Address_A/B, as shown in Fig. 9). This part included two parallel processes of v_{BL} and v_{NP}. Memory vector v_{BL} was stored as a fixed-point number with 9 bits for the whole part and 3 bits for the fractional part. Memory vector v_{NP} was stored as a fixed-point number with 11 bits for the whole part and 3 bits for the fractional part. This choice was made to reduce the FPGA's resource consumption. The fractional part was used to show numbers between zero and one which are made by α and ω.

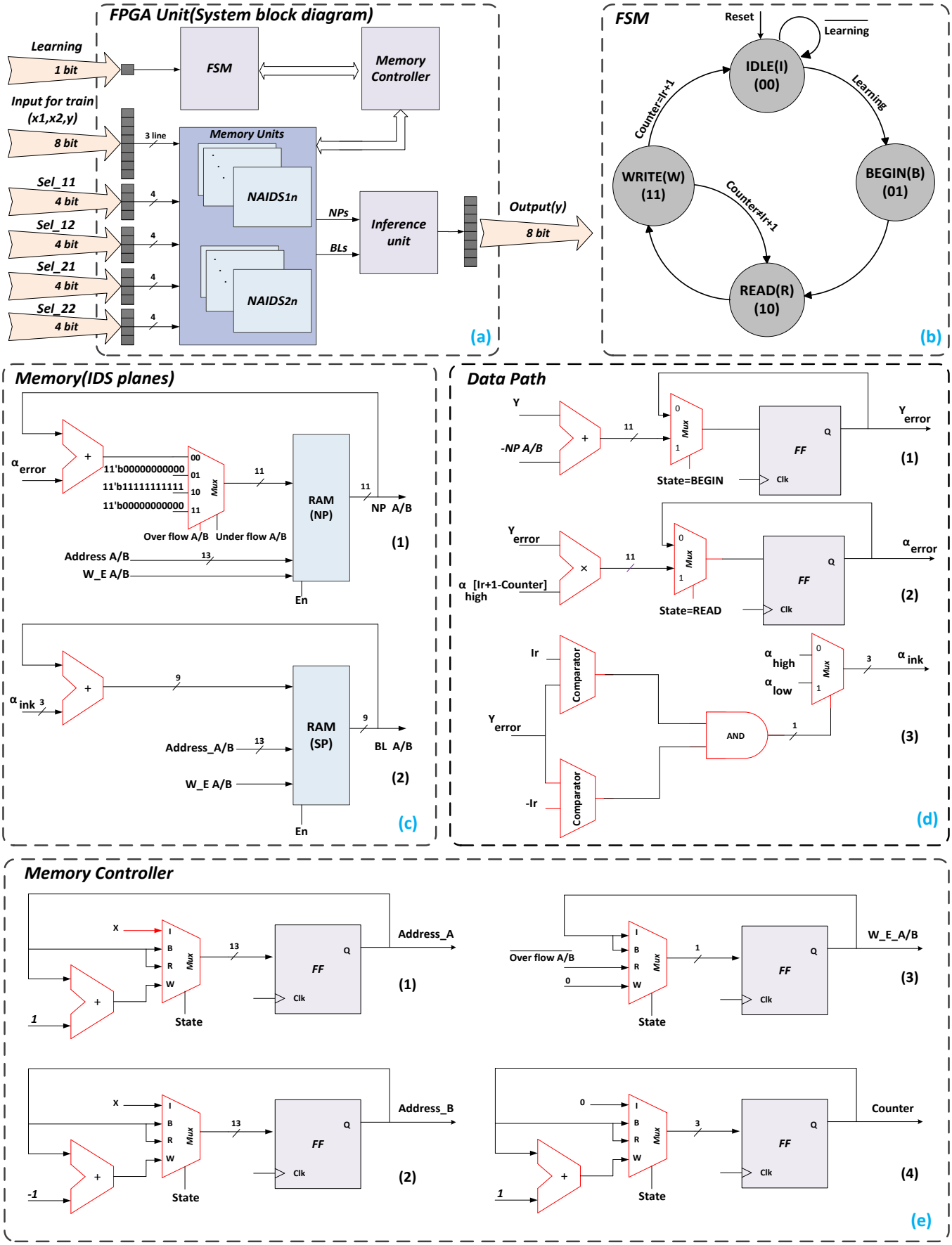

Fig. 9. Structure of the hardware realization of NAALM. (a) Block diagram of the hardware. (b) FSM diagram. (c) Addressing the memory. (d) Data path. (e) Signaling and memory controller.

In Fig. (9-c), the results produced in the next training step were added to the values previously stored in the memory. In the v_{NP} memory vector, the role of the multiplexer in the input of the memory was to feed the output of the adder (final summation result) as input to the RAM block and to compare overflow and underflow. When either of these states occurred, the input would be replaced with, respectively, zero and $256 - 1/8$ (the smallest and largest numbers that can be shown in the fixed-point standard). Since *Belief* was updated with small steps, there was no need for such consideration. The memory length for k planes was $256 \times k$. En was the enable signal for the RAM block and W_E was the enable writing signal. The control logic block was responsible for asserting these enable signals for the RAM block. Address_A and Address_B are the address lines for updating the cells that had to be updated.

As can be seen in Fig. (9-d), the process of updating the value of the memory vectors consisted of three steps. First, in the Begin state, the y value of the input was subtracted and $\mathrm{y_{error}}$ value was computed (in part (d-1)). Secondly, in part (d-2), α_{error} or $\alpha_{high}[\mathrm{Ir} + 1 - \mathrm{counter}] \times \mathrm{y_{error}}$ in the Read state was computed (Ir is the smoothing radius as in (14)). α and g(u) were expressed as a single term $\alpha \times \mathrm{g(u)}$, so to avoid a multiplier, the values of $\alpha \times \mathrm{g(u)}$ for both values of $\alpha_1$and α_2(respectively called $\alpha_{low}$and $\alpha_{high}$with values 0.875 and 0.25) were stored as a 3-bit array with length $\mathrm{Ir} + 1$. Finally, in part (d-3), two comparators and an AND gate checked the condition $|\mathrm{y_{error}}| < \mathrm{Ir}$ for choosing α between α_{high} and α_{low} for use in the *Belief* memory vector. In our hardware implementation, because of the linearity and simplicity of the computing operations, a linear smoothing operator (2-D Pyramid) was used. Since $g(u)$ is symmetric, it was sufficient to store just one half of its value.

The unit shown in Fig. (9-e) was responsible for generating addressing signals. Signals Address_A and Address_B were signals connected to memory ports A and B, which updated two cells of the memory vectors each time by entering the Write state. In part (e-1), first the x value fed into Address_A and in next clock cycle, x added to one and this process controlled by multiplexer (for signal Address_B is the same as part (e-1)). The two ports' write enable signals were activated in the Read state and deactivated in the Write state. In part (e-3), the multiplexer's role was to choose W_E_A/B activation signals for writing. In part (e-4), the counter counted the number of write processes. Because in each update of the planes, $2 * Ir + 1$ cells had to be updated or the dual-port memory had access to up to two cells, for a complete update $Ir + 1$ write processes were required in the memory. The counter counted $Ir + 1$ write processes and then returned to the Idle state. In the Idle state all parameters were returned to their initial values. Each updating needed $Ir + 1$ clocks for reading and $Ir + 1$ clocks for writing, so each learning epoch took $2 * Ir + 2$ clocks. With regard to the symmetry of $g(u)$, by appropriately choosing to update $2 * Ir + 1$ cells of memory (from the middle, in two directions), the number of multipliers could be reduced to a signed 4-bit multiplier and 9-bit multipliers. As shown in the Fig. 9, all the values were stored in Flip Flops (FFs. /Registers).

4.1. Hardware Results

All the materials discussed in the previous section were implemented in serial mode and parallel mode on Spartan-6 FPGA Node-Board, Xilinx Spartan-6 series XC6SLX150T[62] and XC6SLX16 using Verilog synthesis. To demonstrate the proposed method, the software simulation and hardware implementation results are compared in Fig. 10 and Table 5. Fig. (10-a) and Fig. (10-b) show the learned pattern and the convergence of *Narrow Path* and *Belief* for function $\mathrm{Y_1}$ for $\mathrm{X_1} - \mathrm{Y}$ sub-space. As can be seen, the result of the proposed algorithm based on the MATLAB simulation perfectly similar to that obtained in the FPGA implementation, meaning that the hardware implementation was realized perfectly. The comparison of hardware and software results is presented in the Table 5. As can be seen from Table 5, the FPGA implementation of NAALM achieved high precision in approximating functions. According to the results, the speed obtained in the FPGA implementation was so higher than that

obtained in the software implementation. The comparison between software simulation and hardware implementation is reported only for reference and just to show there is no mistake.

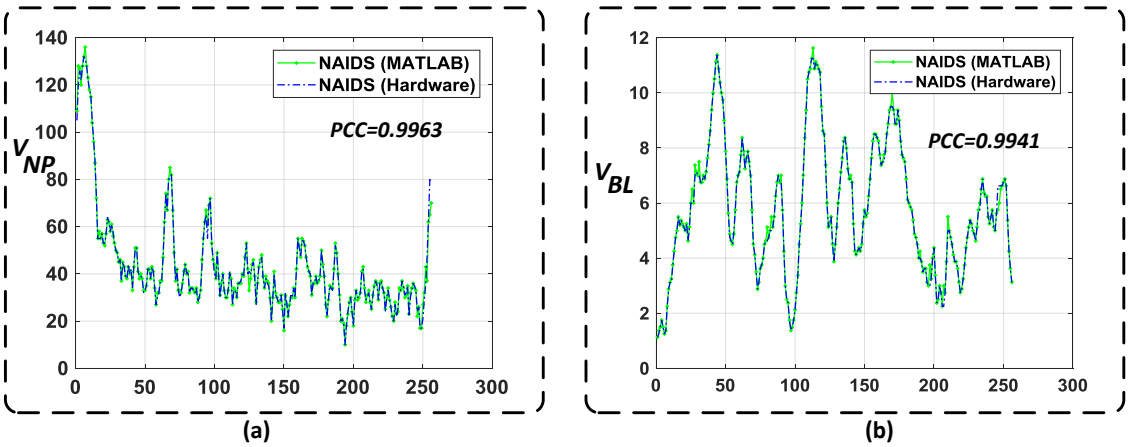

Fig.10. Comparison of proposed algorithm (NAIDS) based on PCC metric, between MATLAB simulation and FPGA implementation in the same condition. (a) Convergence of v_{NP} memory vector. (b) Convergence of v_{BL} memory vector. Training sample size is 1000 and the parameter settings are as follows: $\mathrm{Ir} = 6, \omega = 0.875, \alpha = 0.25$. In this example, there were no fuzzy partitioning points for function Y_1 and $\mathrm{x}_1 - \mathrm{y}$ subspace.

Table 5: Comparison between the results obtained by our proposed hardware in an FPGA parallel implementation with those obtained in a software-based simulation of NAALM based on FVU and PCC metrics. In NAALM, $\alpha = 0.25, \omega = 0.875$, $\mathrm{Ir} = 6$ and The partitioning points in the input domain in Y_1 are 10 for x_1 and x_2, and in Y_2 8 for x_1 and 12 for x_2. The training sample size for all algorithms is 1000. Each algorithm was repeated 20 times and the average reported.

Function	Software simulation (based on MATLAB)			Hardware implementation (based on FPGA XC6SLX150T)		
	FVU	PCC	Time(s)	FVU	PCC	Time(s)
Y_1	0.0278 ± 0.0084	0.9884 ± 0.0067	$0.0681s$	0.0295 ± 0.0071	0.9871 ± 0.0076	$118\,\mu s$
Y_2	0.0335 ± 0.0072	0.9869 ± 0.0073	$0.0719s$	0.0344 ± 0.0069	0.9855 ± 0.0059	$118\,\mu s$

The structure of the proposed algorithm is fully compatible with FPGA constraints. For resource utilization, Table 6 provides resources including (percentage of resources) and the maximum working frequency reported by Xilinx Synthesis Technology (XST) to synthesis the HDL codes on Spartan-6 with timing constraints (Post-place and Route (PAR)). In this table, we compare the resources used in the hardware implementation of the proposed algorithm with those used by other algorithms (other articles) and the results in different families of Xilinx are summarized. Because the amount of resources consumed by the proposed algorithm is very low, for this reason, we compared with small networks of ANFIS and MLP.

As can be seen in Table 6, the proposed algorithm was implemented in serial mode and parallel mode on two devices from the Spartan-6 family. With parallel implementation, a higher speed was achieved, but this also imposed higher resource requirements. We used two kinds of implantation: one with an inference block (defuzzifier subsystem) and one without. We compared the hardware resources needed by other implementations of ANFIS, MLP and with a previous implementation of ALM in many different structures and for various applications.

In Table 6, the second column is the FPGA family and the third column is learning capability, i.e., whether the hardware is learnable or not. When the training phase of the system is on-chip, the proposed architecture does not need to include the training process in software environment, and all the parameters are fixed during system operation. Logic elements were reported. Logic resource utilization included FFs, LUTs, occupied Slices, and RAM blocks for storing memory vectors. For a fair comparison, small MLP and ANFIS networks were chosen, the results showing great optimization in term of resources. In comparison with the other implementations, the proposed hardware used fewer resources. In serial mode, the proposed hardware was able to work with a clock frequency of 156.54 MHz (with a critical path of 6.39 ns), and in parallel mode it was able to work with a clock

frequency of 136.63 MHz (with a critical path of 7.32 ns). This is faster than the previous implementation. The results show that, due to its fast response time, the system is suitable for real-time applications. Apart from its higher speed, the proposed algorithm also has a lower area.

Table 6: The results of hardware realization of NAALM on Xilinx Spartan-6 series, showing. The resources required (utilization) for the proposed algorithm, in both serial and parallel modes. Reported quantities are compared for FPGA implementation of ALM, NAALM, ANFIS, and MLP algorithms. Working frequencies, learning times, occupied slices, etc. are also compared. Using different families is only for indication the low cost of the proposed algorithm.

Algorithm	FPGA Chip	Hardware Learning Capability	Application /Topology	Resources						
				FFs	LUTs	Occupied Slices	BRAM 8/16	Clock Frequency (MHz)	Mean Learning Time Per data	Time for Test Data
Traditional ALM	Cyclone II 2C35F672C6	Yes	1-IDS-Plane	-	795 (≈3%)	-	-	130.13 (7.68 ns)	115.8 ns	-
Pipeline Traditional ALM (10 Stage)		Yes	1-IDS-Plane	-	6805 (23%)	-	-	43.48 (23 ns)	14.48 ns	-
Serial NAALM	Spartan-6 XC6SLX150T	Yes	Function Modeling (20-IDS-Plane)	51 (<1%)	158 (<1%)	57 (<1%)	9 RAM 16B (3%)	156.54 (6.39 ns)	408.8 ns	6.39 ns
Serial NAALM with defuzzification		Yes	Function Modeling (20-IDS-Plane)	92 (<1%)	665 (<1%)	99 (<1%)	9 RAM 16B (3%)	145.92 (6.85)	438.6 ns	68.5 ns
Parallel NAALM		Yes	Function Modeling (20-IDS-Plane)	376 (<1%)	2572 (2.8%)	920 (4%)	40 RAM 8B (7%)	136.63 (7.32 ns)	117.1 ns	7.32 ns
parallel NAALM with defuzzification		Yes	Function Modeling (20-IDS-Plane)	384 (<1%)	3061 (3.3%)	1146 (5%)	40 RAM 8B (7%)	135.78 (7.37 ns)	117.8 ns	59.0 ns
Serial NAALM	Spartan-6 XC6SLX16	Yes	Function Modeling (20-IDS-Plane)	51 (<1%)	158 (1.7%)	66 (2.9%)	9 RAM 16B(28%)	151.19 (6.61 ns)	423.3 ns	6.61 ns
Serial NAALM with defuzzification		Yes	Function Modeling (20-IDS-Plane)	92 (<1%)	665 (7.3%)	248 (11%)	9 RAM 16B(28%)	139.53 (7.17 ns)	458.7 ns	71.7 ns
Parallel NAALM		Yes	Function Modeling (20-IDS-Plane)	376 (2.1%)	2563 (28%)	911 (40%)	40 RAM 8B (63%)	132.77 (7.53 ns)	120.5 ns	7.53 ns
Parallel NAALM with defuzzification		Yes	Function Modeling (20-IDS-Plane)	383 (2.1%)	3062 (34%)	1107 (49%)	40 RAM 8B (63%)	130.29 (7.68 ns)	122.9 ns	53.7 ns
ANFIS in [63]	Spartan-3 XC3S200	No	Function Modeling (3 MFs in variables)	881 (22%)	2042 (53%)	1100 (57%)	-	50 (20 ns)	-	15 μs
ANFIS in [21]	Spartan-3 XC3S200	No	Function Modeling (3 MFs in variables)	674 (17%)	1457 (37%)	762 (39%)	-	50 (20 ns)	-	8 μs
ANFIS in [64]	Spartan-6 XC6SL45	No	Controller (3 MFs in variables)	-	936 (3%)	446 (6%)	21 RAM 16B(17%)	-	-	75 ns
MLP in [65]	Virtex 2 pro	Yes	XOR problem (1-2-1)	544 (2%)	635 (2.3 %)	578 (4.2%)	2 RAM 16B(2%)	82.96 (12 ns)	-	-
MLP in [66]	Virtex 2 X2V250FG	Yes	XOR problem (2-6-2-2)	846 (27%)	1931 (55%)	1299 (84%)	8 RAM 16B(33%)	41.55 (24 ns)	0.78 μs	0.18 μs
MLP in [22]	Virtex-5 Xc5vsx50t	Yes	Spectrometry (10-3-1)	2243 (6%)	8043 (24 %)	489 (6%)	-	-	-	-
MLP in [29]	Virtex XCV400hq240	No	Estimator (8-5-5-3)	2558 (26%)	5460 (56 %)	2993 (62%)	2 RAM (20 %)	73 (13.69 ns)	-	-
MLP in [67]	Spartan-6 AC6SLX150T	No	Classification (9-9-6)	4239 (2.3 %)	2226 (2.4 %)	-	34 RAM 8B(6 %)	100 (10 ns)	-	1 μs

As can be seen, the proposed hardware reduced the time required for training compared to previous hardware solutions. In the training phase, the “Mean learning time per data” is the time the hardware needs to update the block RAMs. When the training phase is complete, in the test phase, the “Time for test data” is the time needed to read the memory vectors and the available output for the input test data. As can be seen in the table, the hardware implementation of the proposed algorithm was faster than that of all the other algorithms because it required no complex computation.

The MLP and ANFIS training process was based on least square methods, which are computationally expensive. Both MLP and ANFIS are multiplication rich by nature, and they use a lot of FPGA resources. As can be seen, the ANFIS implementation is not on-chip training and only some of the MLP implementation are on-chip training. This is fine for solving simple problems (in small neural networks) but is unusable for real life solutions. In contrast, the proposed algorithm is on-chip training and online learning, because it is non-iterative and has less hardware. In fact, the algorithm has simple computations, such as summation, subtraction, and multiplication.

Our proposed algorithm eliminates some of the drawbacks that afflict traditional ALM implementations on VLSI circuits. This novel digital structure has more flexibility and scalability than previous architectures and other algorithms. To date, almost all hardware that has been proposed has been offline, and training phase has been conducted in software, with the results being implemented on hardware. Consequently, one of the most important advantages of the proposed algorithm (the main contribution of the paper) is that its hardware implementation is capable of on-chip training for applications with time variant parameters. Regardless of the application, the only thing that has to be changed is the coefficient. There is no need to modify the structure (it is easily scaled). Learnable hardware is essential for future intelligent systems. Ambient intelligence needs small embedded systems (hardware implementation) able to learn dataset online with a large number of inputs, and also able to adapt themselves to changing conditions.

5. CONCLUSION

ALM is a computing tool based on brain simulation. Despite successful applications of this algorithm in various domains, its IDS operations suffer from high computational costs. In this paper, a novel learning algorithm is proposed as an alternative to the original method. The proposed algorithm makes use of describing vectors (memory vectors) for the IDS planes. This modification has resulted not only in an acceptable level of performance, but also in considerable reduction of the hardware area. Different simulations on real-world datasets showed that the proposed method is not only very effective, simple and fast, but also just as accurate as other algorithms. In the FPGA implementations (the proposed hardware is described directly using Hardware Description Language (HDL)) carried out as part of this study, the following criteria were taken into account; learning speed, throughput, computational independence, memory requirement, output precision, element limitations, and scalability and flexibility in an adaptive systems. Therefore, the proposed algorithm is the implementation of a complete fuzzy systems and it can be implemented in any commercial FPGA or ASIC technology. The main advantage of the proposed algorithm and its hardware is its online training capability (on-chip learning), which makes it suitable for time variant and real-time systems (due to its fast response time, it has no limitations with regard to applications). In NAALM, inputs and output are fuzzy and in the training phase, the vectors are updated in the same way as in the SOM (Self Organizing Map) algorithm described in (14). In addition, by assuming an individual neuron for each quantization level, each input reinforces the weights of a set of neurons, as happens when training SNNs. Ultimately; the proposed method can be considered as a neuro-fuzzy system linking up the two domains of ANNs and fuzzy systems. The proposed algorithm scales well in dataset with very large number of samples due to low memory requirement and computational nature. Unlike other algorithms, in dealing with datasets that require more neurons to increase accuracy, the proposed method is able to easily learn these datasets and does not require increasing hardware volume.

However, in dataset with large number of features, NAALM (like ALM) suffers from the complexity. In this regard, further research is required to find a novel approach (using a matrix memory instead of using memory vectors in learning phase or proposing an algorithm that does not need to split system). The MATLAB codes and Verilog codes of the FPGA circuits used are available as supplementary material for download.

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

Acknowledgments

The authors would like to thank Mohsen Firouzi and Menoa Keshishian for their generous contribution to our analysis. This work was partially supported by the INSF (Iran National Science Foundation) grant number 96000943.

REFERENCES

[1] Tsoukalas, L.H. and R.E. Uhrig, *Fuzzy and neural approaches in engineering*. 1996: John Wiley & Sons, Inc.
[2] Samarasinghe, S., *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. 2006: CRC Press.
[3] Zadeh, L.A., *Fuzzy sets.* Information and control, 1965. 8(3): p. 338-353.
[4] Zadeh, L.A., *Outline of a new approach to the analysis of complex systems and decision processes.* IEEE Transactions on systems, Man, and Cybernetics, 1973(1): p. 28-44.
[5] Widrow, B., D.E. Rumelhart, and M.A. Lehr, *Neural networks: applications in industry, business and science.* Communications of the ACM, 1994. 37(3): p. 93-106.
[6] Basheer, I. and M. Hajmeer, *Artificial neural networks: fundamentals, computing, design, and application.* Journal of microbiological methods, 2000. 43(1): p. 3-31.
[7] Ukil, A., *Intelligent systems and signal processing in power engineering*. 2007: Springer Science & Business Media.
[8] Paliwal, M. and U.A. Kumar, *Neural networks and statistical techniques: A review of applications.* Expert systems with applications, 2009. 36(1): p. 2-17.
[9] Haykin, S., *Neural Networks: A Comprehensive Foundation: Macmillan College Publishing Company*. New York, 1994.
[10] Park, B.-J., W. Pedrycz, and S.-K. Oh, *Polynomial-based radial basis function neural networks (P-RBF NNs) and their application to pattern classification.* Applied Intelligence, 2010. 32(1): p. 27-46.
[11] Sun, X.-y., et al., *Improved probabilistic neural network PNN and its application to defect recognition in rock bolts.* International Journal of Machine Learning and Cybernetics, 2016. 7(5): p. 909-919.
[12] Bezdek, J.C., *Fuzzy mathematics in pattern classification*. 1973.
[13] Mamdani, E.H., *Application of fuzzy algorithms for control of simple dynamic plant.* Electrical Engineers, Proceedings of the Institution of, 1974. 121(12): p. 1585-1588.
[14] Takagi, T. and M. Sugeno, *Fuzzy identification of systems and its applications to modeling and control.* IEEE transactions on systems, man, and cybernetics, 1985(1): p. 116-132.
[15] Tanaka, K. and H.O. Wang, *Fuzzy control systems design and analysis: a linear matrix inequality approach*. 2004: John Wiley & Sons.
[16] Terano, T., K. Asai, and M. Sugeno, *Applied fuzzy systems*. 2014: Academic Press.
[17] Bosque, G., I. del Campo, and J. Echanobe, *Fuzzy systems, neural networks and neuro-fuzzy systems: A vision on their hardware implementation and platforms over two decades.* Engineering Applications of Artificial Intelligence, 2014. 32: p. 283-331.
[18] Singh, P., *A brief review of modeling approaches based on fuzzy time series.* International Journal of Machine Learning and Cybernetics, 2017. 8(2): p. 397-420.
[19] Misra, J. and I. Saha, *Artificial neural networks in hardware: A survey of two decades of progress.* Neurocomputing, 2010. 74(1): p. 239-255.
[20] Monmasson, E., et al., *FPGAs in industrial control applications.* IEEE Transactions on Industrial informatics, 2011. 7(2): p. 224-243.
[21] Saldaña, H.J.B. and C.S. Cárdenas. *Design and implementation of an adaptive neuro-fuzzy inference system on an FPGA used for nonlinear function generation*. in *ANDESCON, 2010 IEEE*. 2010.
[22] Gomperts, A., A. Ukil, and F. Zurfluh, *Development and implementation of parameterized FPGA-based general purpose neural networks for online applications.* IEEE Transactions on Industrial Informatics, 2011. 7(1): p. 78-89.
[23] Eldredge, J.G. and B.L. Hutchings. *RRANN: a hardware implementation of the backpropagation algorithm using reconfigurable FPGAs*. in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*. 1994.
[24] Yun, S.B., et al. *Hardware implementation of neural network with expansible and reconfigurable architecture*. in *Neural Information Processing, 2002. ICONIP'02. Proceedings of the 9th International Conference on*. 2002. IEEE.
[25] del Campo, I., et al., *Efficient hardware/software implementation of an adaptive neuro-fuzzy system.* IEEE Transactions on Fuzzy Systems, 2008. 16(3): p. 761-778.
[26] Baptista, F.D. and F. Morgado-Dias, *Automatic general-purpose neural hardware generator.* Neural Computing and Applications, 2017. 28(1): p. 25-36.
[27] Lacey, G.J., *Deep Learning on FPGAs*. 2016.
[28] Ortega-Zamorano, F., et al., *Layer multiplexing FPGA implementation for deep back-propagation learning.* Integrated Computer-Aided Engineering, 2017. 24(2): p. 171-185.
[29] Himavathi, S., D. Anitha, and A. Muthuramalingam, *Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization.* IEEE Transactions on Neural Networks, 2007. 18(3): p. 880-888.
[30] Ortega-Zamorano, F., et al., *Efficient implementation of the backpropagation algorithm in fpgas and microcontrollers.* IEEE transactions on neural networks and learning systems, 2016. 27(9): p. 1840-1850.

[31] Soudry, D., et al., *Memristor-based multilayer neural networks with online gradient descent training.* IEEE transactions on neural networks and learning systems, 2015. 26(10): p. 2408-2421.

[32] Ortigosa, E.M., et al., *Hardware description of multi-layer perceptrons with different abstraction levels.* Microprocessors and Microsystems, 2006. 30(7): p. 435-444.

[33] Shouraki, S.B., *A novel fuzzy approach to modeling and control and its hardware implementation based on brain functionality and specifications.* 2000.

[34] Sugeno, M. and T. Yasukawa, *A fuzzy-logic-based approach to qualitative modeling.* IEEE Transactions on fuzzy systems, 1993. 1(1): p. 7-31.

[35] Murakami, M. and N. Honda, *A study on the modeling ability of the IDS method: A soft computing technique using pattern-based information processing.* International journal of approximate reasoning, 2007. 45(3): p. 470-487.

[36] Bahrpeyma, F., A. Zakerolhoseini, and H. Haghighi, *Using IDS fitted Q to develop a real-time adaptive controller for dynamic resource provisioning in Cloud's virtualized environment.* Applied Soft Computing, 2015. 26: p. 285-298.

[37] Sakurai, Y., *A study of the learning control method using PBALM-a nonlinear modeling method.* PhD, The University of Electro-Communications, Tokyo, 2005.

[38] Shahdi, S.A. and S.B. Shouraki. *Supervised active learning method as an intelligent linguistic controller and its hardware implementation*. in *2nd IASTEAD International Conference on Artificial Intelligence and Applications (AIA'02), Malaga, Spain*. 2002.

[39] Shouraki, S.B. and N. Honda. *Fuzzy controller design by an active learning method*. in *31th Symposium of Intelligent Control. Tokyo, Japan*. 1998.

[40] MURAKAMI, M., *Practicality of modeling systems using the IDS method: Performance investigation and hardware implementation*. 2008, The University of Electro-Communications.

[41] Firouzi, M., S.B. Shouraki, and J. Conradt. *Sensorimotor Control Learning Using a New Adaptive Spiking Neuro-Fuzzy Machine, Spike-IDS and STDP.* in *International Conference on Artificial Neural Networks*. 2014. Springer.

[42] Cranganu, C. and F. Bahrpeyma, *Use of active learning method to determine the presence and estimate the magnitude of abnormally pressured fluid zones: a case study from the Anadarko Basin, Oklahoma,* in *Artificial Intelligent Approaches in Petroleum Geosciences*. 2015, Springer. p. 191-208.

[43] Merrikh-Bayat, F., F. Merrikh-Bayat, and S.B. Shouraki, *The neuro-fuzzy computing system with the capacity of implementation on a memristor crossbar and optimization-free hardware training.* IEEE Transactions on Fuzzy Systems, 2014. 22(5): p. 1272-1287.

[44] Ghorbani, M.J., M.A. Choudhry, and A. Feliachi. *Distributed multi-agent based load shedding in power distribution systems*. in *Electrical and Computer Engineering (CCECE), 2014 IEEE 27th Canadian Conference on*. 2014. IEEE.

[45] Shouraki, S.B., N. Honda, and G. Yuasa, *Fuzzy interpretation of human intelligence.* International Journal of Uncertainty, Fuzziness and Knowledge-based Systems, 1999. 7(04): p. 407-414.

[46] Firouzi, M., S.B. Shouraki, and I.E.P. Afrakoti, *Pattern analysis by active learning method classifier.* Journal of Intelligent & Fuzzy Systems, 2014. 26(1): p. 49-62.

[47] Javadian, M., S.B. Shouraki, and S.S. Kourabbaslou, *A novel density-based fuzzy clustering algorithm for low dimensional feature space.* Fuzzy Sets and Systems, 2016.

[48] Klidbary, S.H., et al. *Outlier robust fuzzy active learning method (ALM). In: IEEE 2017, 7th International Conference on Computer and Knowledge Engineering (ICCKE)*, pp. 347-352.

[49] Shahraiyni, T.H., et al., *Application of the Active Learning Method for the estimation of geophysical variables in the Caspian Sea from satellite ocean colour observations.* International Journal of Remote Sensing, 2007. 28(20): p. 4677-4683.

[50] Sagha, H., et al. *Real-Time IDS using reinforcement learning*. in *Intelligent Information Technology Application, 2008. IITA'08. Second International Symposium on*. 2008. IEEE.

[51] Merrikh-Bayat, F., S.B. Shouraki, and A. Rohani, *Memristor crossbar-based hardware implementation of the IDS method.* IEEE Transactions on Fuzzy Systems, 2011. 19(6): p. 1083-1096.

[52] Shouraki, S.B., *A novel fuzzy approach to modeling and control and its hardware implementation based on brain functionality and specifications.* 2000.

[53] Shouraki, S.B. and N. Honda, *Recursive Fuzzy Modeling Based on Fuzzy Interpolation.* JACIII, 1999. 3(2): p. 114-125.

[54] Shouraki, S.B. and N. Honda. *Outlines of a soft computer for brain simulation*. in *International Conference on Soft Computing Information/Intelligence Systems*. 1998.

[55] Bagheri, S. and N. Honda. *Hardware simulation of brain learning process*. in *15 Fuzzy Symposium*. 1999.

[56] Murakami, M. and N. Honda. *Hardware for a new fuzzy-based modeling system and its redundancy*. in *Fuzzy Information, 2004. Processing NAFIPS'04. IEEE Annual Meeting of the*. 2004. IEEE.

[57] Tarkhan, M., S.B. Shouraki, and S.H. Khasteh, *A novel hardware implementation of IDS method.* IEICE Electronics Express, 2009. 6(23): p. 1626-1630.

[58] Rabaey, J.M., A.P. Chandrakasan, and B. Nikolic, *Digital integrated circuits*. Vol. 2. 2002: Prentice hall Englewood Cliffs.

[59] Firouzi, M., et al. *A novel pipeline architecture of Replacing Ink Drop Spread*. in *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*. 2010. IEEE.

[60] Afrakoti, I.E.P., S.B. Shouraki, and B. Haghighat, *An Optimal Hardware Implementation for Active Learning Method Based on Memristor Crossbar Structures.* IEEE Systems Journal, 2014. 8(4): p. 1190-1199.

[61] Afrakoti, I.E.P., A. Ghaffari, and S.B. Shouraki. *Effective partitioning of input domains for ALM algorithm*. in *Pattern Recognition and Image Analysis (PRIA), 2013 First Iranian Conference on*. 2013. IEEE.

[62] Iakymchuk, T., et al. *An AER handshake-less modular infrastructure PCB with x8 2.5 Gbps LVDS serial links*. in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. 2014. IEEE.

[63] Saldaña, H.J.B. and C. Silva-Cárdenas. *A digital hardware architecture for a three-input one-output zero-order ANFIS*. in *Circuits and Systems (LASCAS), 2012 IEEE Third Latin American Symposium on*. 2012. IEEE.

[64] Gómez-Castañeda, F., et al. *Photovoltaic panel emulator in FPGA technology using ANFIS approach*. in *Electrical Engineering, Computing Science and Automatic Control (CCE), 2014 11th International Conference on*. 2014. IEEE.

[65] Bahoura, M. and C.-W. Park. *FPGA-implementation of high-speed MLP neural network*. in *Electronics, Circuits and Systems (ICECS), 2011 18th IEEE International Conference on*. 2011. IEEE.

[66] Gironés, R.G., et al., *FPGA implementation of a pipelined on-line backpropagation.* Journal of VLSI signal processing systems for signal, image and video technology, 2005. 40(2): p. 189-213.

[67] Echanobe, J., R. Finker, and I. del Campo. *A divide-and-conquer strategie for FPGA implementations of large MLP-based classifiers*. in *Neural Networks (IJCNN), 2015 International Joint Conference on*. 2015. IEEE.

Appendix A. Proof of convergence

As it has also been discussed, the training mode in the original ALM algorithm is batch-Mode, and in NAALM is Sample-Mode. In order to prove the convergence of v_{NP} vector, first, we show the change in one element of this vector for training samples:

$$v^I_{NP_{K_I+1}} = v^I_{NP_0} + \omega \times \sum_{t=0}^{K_I} e^{-\frac{(I-x_t)^2}{2\sigma^2}} \times (y_t - v^{x_t}_{NP_t}) \quad \text{(A .1)}$$

Where, (x_t, y_t) is training sample, I is the index of the vector for updating, K_I is the total number of training samples in element I, ω is learning rate, x_t is the input that vary between one and the quantization level of X axis (L_x). For each training sample associated with each quantization level, following equation holds:

$$k_1 + k_2 + \cdots + k_{L_x} = K \quad \text{(A . 2)}$$

K is the total number of training samples (number of iterations) in the IDS plane. For the benefit of simplicity and less computations, if $x_t = I$, the previous equation can be simplified:

$$v^I_{NP_{K_I+1}} = v^I_{NP_0} + \omega \times \sum_{t=0}^{K_I} (y_t - v^I_{NP_t}) \quad \text{(A . 3)}$$

If we expand the summation, a recursive equation can be obtained as follows:

$$v^I_{NP_{K_I+1}} = (1-\omega)^{k_I+1} \times v^I_{NP_0} + \omega \times \sum_{i=0}^{K_I} (1-\omega)^{k_I-i} * y_i \quad \text{(A . 4)}$$

If $K_I \gg 1$, in the previous equation, the first term of the equation tends to zero, and by assuming $m = k_I - i$, following equation can be obtained:

$$v^I_{NP_{K_I+1}} = \omega \times \sum_{i=0}^{m} (1-\omega)^m * y_{k_I-m} \quad \text{(A . 5)}$$

This equation shows that the final output of the algorithm converges to the weighted sum of outputs. With regard to forgetting property introduced in the proposed method (NAALM), the impact of training samples that were shown later in the training phase is higher than that of training samples that were shown to the algorithm earlier. It should be mentioned that because of the fact that NAALM algorithm partition the inputs domains. Therefore, for the mentioned recursive algorithm, the standard deviation of v_{NP} is small and because the computations are conducted in the fuzzy space (space with uncertainty), the lack of equivalency (the difference between Batch-Mode and Sample-Mode in training phase) do not significantly affect the final result.

With regard to assumptions we made about the convergence of describing vectors, the value of degree of Belief would be:

$$v^I_{BL_{K_I+1}} = v^I_{BL_0} + \alpha \quad \text{(A . 6)}$$

If we expand the summation, a recursive equation can be obtained as follows:

$$v^I_{BL_{K_I+1}} = v^I_{BL_0} + \sum_{i=0}^{K_I} \alpha \quad \text{(A . 7)}$$

If $K_I \gg 1$, in the previous equation, following equation can be obtained:

$$v^I_{BL_{K_I+1}} = v^I_{BL_0} + (K_I + 1) \times \alpha \quad \text{(A . 8)}$$

This equation shows that as the number of training samples (or the number of iterations) increases, the value of Belief increases, and eventually our belief to the occurrence of that event converge to constant value ($K_{\mathrm{I}} < \infty$ and $\alpha < 1$).