Bidirectional Associative Memories for Adaptive Rule Based Systems

Nabil M. Hewahi
Department of Computer Science
College of Information Technology, University of Bahrain
Bahrain International Circuit, Zallaq, Bahrain
Phone: +973 1743 8888
nhewahi@uob.edu.bh

Abstract

In this paper we present an algorithm for rule based systems that uses Bidirectional Associative Memories (BAMs) to memorize the inputs and their corresponding outputs, and outputs and their corresponding inputs of the system. Adaptive rule based system are becoming very important because rule based systems are now used in many applications. One drawback of current adaptive rule based systems is that these systems care only about forward chaining mechanism which diminish their performance. Because most of the applications that use rule based systems follow the forward chaining, adaption attempts which is concerned with predicting the inputs given the output is almost null, but this does not eliminate the importance of backward chaining since it is used in too many applications. To tackle this problem we propose a new algorithm that utilizes the good theoretical ground of BAMs to memorize and adapt rules in rule based systems. The main difference between the proposed algorithm and other algorithms is that the proposed algorithm is simple to code the rules and adapt them. In addition, because BAMs supports bi-directions, the proposed algorithm is the only algorithm with adaptation that is able to expect what conditions should be true if we provide the system with outputs. The proposed algorithm considers "and" and "or" rules in the used rules, whereas in all the previous systems, only "and" relation is considered. The proposed solution will be useful in adapting any rule based system whether it uses forward chaining or backward chaining, which is considered to be a significant contribution. The proposed algorithm has six parts; cod table creation, input and output vectors construction, weight matrix calculation, procedure to test the system, procedure to run the proposed algorithm and finally rule extraction or what we call rule code decoding. To illustrate parts of the proposed algorithm, a simple example is provided.

Keywords: Rule Based Systems; Adaptive Systems; Bidirectional Associative Memories.

1. Introduction

In this section, we shall give an overview on two main issues related to the basics of the work of this paper, rule based systems and adaptation; and BAMs.

1.1. Rule Based Systems and Adaptation

Rule based systems are systems that work based on given rules related to the problem domain. Most of these systems use standard rule structure, where each rule is having conditions and actions. The conditions might be connected with "and" or "or" relation. To obtain the action, the condition has to be true. The standard rule structure is having the form:

IF Conditions/ Antecedents THEN Actions/Consequences

Usually rule based system uses an inference mechanism to obtain the final conclusion, this inference could be forward chaining, backward chaining or both together. In forward chaining inputs for the conditions are given to get the outputs, on the other hand, during the backward chaining, the expected output is given to the system and the then system goes backward to prove it if possible. The concepts of forward chaining and backward chaining are very important in creating adaptive rule based systems. Many trails have been made to develop adaptive rule based systems, and most of these trails depend on firstly representing the rules in a certain approach then train the

system to be able to induce new rules (Blount et al., 2011; Cimino et al., 2012; Farid et al., 2016; Hewahi, 2004; Hewahi, 2005; Hewahi, 2011). In addition, these approaches consider only forward chaining mechanism, and are usually complex because they need a sophisticated rule representation stage before learning. Additionally, in the previous attempts, only "and" relation between the conditions is considered. The advantages of these approaches are that new rules might be extracted.

Because many rule based systems depend on backward chaining or both forward and backward chaining, it would be necessary to have adaptive systems that consider such kind of chaining. These adaptive systems should consider simple representation of rules and deal with "and" and "or" relations within the condition part of the rule.

In our proposed approach, rule representation and the learning process are very simple processes. In addition, rules can be extracted. And above all, this approach can work in forward chaining as well as in backward chaining considering also rules with "and" and "or" condition relations. The proposed approach depends on BAMs that have been used as a theoretical ground to memorize and adapt the rules, and to facilitate working with forward and backward chaining in the adaptive rule based system.

In this introduction, we present two subsections related to the core work of this paper. The first part is to give an overview on rule based systems and rule structure. The second part is to go through BAMs.

1.2. BAMs

BAMs are neural networks proposed by Kosko (1987) and works in bidirectional. BAMs is an extension of Hopfield networks which works in one direction (1982). One direction means the neural network can memorize its inputs. In Hopfield networks, the networks memorize the given inputs. Bidirectional means the neural networks can memorize outputs related to given certain inputs, and can also memorize inputs related to given outputs. BAMs are two layer neural networks whereas the Hopfield network is a one layer neural network. The advantages of BAMs over Hopfield networks is that when input is given, we can get the corresponding output, and when the output is given, we can know the corresponding input. In BAMs, we need first to calculate the weight matrix W which is defined in formula (1)

$$W = \sum_{i=1}^{n} (X_i . Y_i^t) \qquad (1)$$

Where n is the number of examples, X_i is the i^{th} example and Y_i^t is the transposed output corresponding to X_i. The inputs values for X_i is 1 for true input and -1 for false input. Similarly for output values.

To test the network, we can give any input from the given input examples, or new input not seen before. In case, the input is from the given examples, the system should produce the corresponding output, and in case the input is not given before, the system is expected to get one of the given outputs; however, if obtained output is not among the previously given outputs, we can still get the most reasonable expected output. Similarly, if the output is given as in the memorizing examples, the system will be able to obtain the necessary inputs. If the given output is not one of the memorizing examples, the system will be able to give us the closest required input. Formula (2) is used to compute the output vector

$$F(W^t X_i) \qquad (2)$$

Assuming $Z = W^t X_i$, F will give value 1 if $Z>0$ and will give value -1 if $Z< 0$. If $Z = 0$, F will get value of the last output 1 or -1.

Assume we have two inputs X_1 and X_2 and two outputs Y_1 and Y_2 (where Y_1 is the output for X_1 and Y_2 is the output for X_2) as below:

$X_1 = [\ 1 \quad 1]$ $\quad Y_1 = [1 \quad -1 \quad -1]$

$X_2 = [-1 \quad 1]$ $\quad Y_2 = [1 \quad 1 \quad -1]$

$W = X_1.Y_1^t + X_2.Y_2^t$

$$W = \begin{pmatrix} 0 & -2 & 0 \\ 2 & 0 & -2 \end{pmatrix}$$

Let us now assume that we want to test the network with X_1 as input.

$$F\left(\begin{pmatrix} 0 & 2 \\ -2 & 0 \\ 0 & -2 \end{pmatrix} . \begin{pmatrix} 1 \\ 1 \end{pmatrix}\right)$$

$= [\ 1 \quad -1 \quad -1]^T$

If we now want to see the association between the output and input, assume

$[1 \quad -1 \quad -1]^T$ as input

$= F(WX_i)$

$$= F\left(\begin{pmatrix} 0 & -2 & 0 \\ 2 & 0 & -2 \end{pmatrix} . \begin{pmatrix} 1 \\ -1 \\ -1 \end{pmatrix}\right)$$

$$= F\left([2 \quad 4]^T\right)$$

$= [\ 1 \quad 1]^T$

Now let us assume we want to know the expected output for the input $[1 \quad -1]$, following the same procedure, the expected output will be

$[-1 \quad -1 \quad 1]^T$

And if want to see the association from output to input and start from the output $[-1 \quad -1 \quad 1]^T$, we get $[1 \quad -1]$ as inputs.

2. Literature Review

Many attempts to develop adaptive rule based systems have been done. Most of these attempts are firstly focus on representing the existing rule set in a certain structure that can be used for learning process. Holland (1985) developed a system called a classifier system that learns classifiers (classifiers are rules coded into bit patterns). The system uses an algorithm called bucket

brigade algorithm to make conflict resolution between classifiers and to assign credit to each classifier. New classifiers are generated based on genetic algorithms. The main disadvantage of this process is the long action chain caused by the bucket brigade algorithm. Moreover, this approach has nothing to do with backward chaining and only "and" relation is considered between the conditions. In (Towell & Shavlik, 1991; Towell & Shavlik, 1994), a proposal called Knowledge Based Neural Network (KBNN) has been presented. The idea is to represent the rules first in a neural network and consider important inputs with high weights and unseen inputs with very low weights. After training, some of the weak weights might become strong and new rules can be constructed. Again, in this process backward chaining is not considered and only "and" relation is used between the conditions. Bahradwaj and Silva (1998) proposed an approach to develop an adaptive system for hierarchical censored production rules (a special type of a rule structure that handles real time cases) has been proposed. This approach is called variable precision neural logic. The used approach is a sort of integration between hierarchical censored production rules and neural networks. Actually this approach also takes a quite some time to represent the rules first. In addition it does not deal with backward chaining despite the used rule structure facilitates to go backward. Also "or" relation between the conditions is omitted. Hewahi (2004) introduced a new method called General Rule Structure Neural Logic (GRSNL) Network. This approach was developed to have an adaptive system for rules called General Rule Structure (GRS) rule (Hewahi, 2002). This approach also needs to represent the rules first into a neural network, and then training is performed to extract new rules. Again this approach cares only about forward chaining and considers only "and" relation between the conditions. In (Hewahi, 2011) an approach based on probabilistic neural networks has been presented. In this method rules are first implemented in a probabilistic neural network and then the neural network can answer queries. This approach also needs full representation and implementation of all the given rules into the probabilistic neural networks. This approach is only useful for forward chaining. Wang (2017) claims that there is no algorithm to adapt business rules that are used in industry when known information is in the form of statistical nature. He proposed an algorithm based on mathematical programming. The scope of the proposed algorithm was based on two concepts which are called bounded case and linear iteration bounded. This approach is still limited to business and the information of statistical nature and the main concern is only the adaption. Tabebordbar and Beheshti (2018) proposed an adaptive monitoring system to monitor the precision of the rules. Their approach can recognize rules that cause the system to perform bad, the proposed approach could catch the improper rules. Actually, this work has nothing to do directly with the work in this paper, but still it shows the importance of rule based systems and their adaptation.

Many adaptive rule based systems have been developed for certain areas such as Alzheimer classification (Jain et al, 2013), facial expression recognition (Ioannou et al, 2004), managing situation-awareness (Cimino et al, 2012) , classifier for mining big biological data (Farid et al, 2016), classifying activities of daily living (Okour et al, 2015), malware detection (Blount et al, 2011), and smart home energy prediction (Jithish & Sankaran, 2017) In all the previous works, the used approach is only designed to fit with the used application domain, which means any approach used in one work is not directly useful to other domains.

To sum up, all the previous works for developing adaptive rule based systems need to represent rules first and then start training. In addition, backward chaining is not considered. In case of designing adaptive rules based systems for specific domains, these approaches do not work with other domains and would be difficult to work as a general approach.

In our paper we present an approach that does not need to represent the rules in a complex structure and avoid the drawbacks of the previous methods. We depend on the theory of BAMs which deals with input and output. The process which we follow can memorize the inputs and their corresponding outputs, and outputs and their corresponding inputs. In backward chaining we check whether the goal can be achieved starting from the goal, but in our approach, going backward means known what are the necessary conditions need to be true to achieve the goal. The proposed approach can be used in various application domains. The advantage of using BAMs for adaptive

rule based systems is that the most of rule based systems have a limited number of rules that can be coded and represented easily.

3. The Proposed Approach

As stated above our proposed approach depends on BAMs Theory. We assume rules have only one condition, "and" relation conditions or "or" relation conditions, but "and" and "or" relations are not considered in one rule. In most of the previous approaches, only one condition or "and" relation conditions are only considered, but "or" relation is not considered. Also our assumptions include that any rule has only one conclusion, positive or negative, which is also the same in all the previous algorithms. The proposed approach has five parts, creation of coding table, input and output vectors construction, obtaining the weight matrix, testing the system, running the system and rule extraction (rule decoding).

3.1. Coding Table

In this stage, rules are coded in a specific table called coding table. Coding table contains the rules condition inputs, the rules outputs and whether the relation in the rule condition is "and" relation or "or" relation. Each row in the table represents a rule and is called rule code. Each rule input slot is filled with 1 if true and -1 if false, and each rule output slot corresponding to the rule is filled with 1 if positive (true) and -1 if negative (false). Under the relation column, if the rule is having "and" relation, its relation slot is filled with 0, if the rule is having "or" relation, its relation slot is filled with 1. If the rule has only one condition, the slot becomes empty. The used table is shown in Table 1.

Table 1. The proposed rules coding table for three rules. The total number of inputs in all rules is 8 and total number of outputs in all rules is 3

	Inputs								Relation	Outputs		
#	A	B	C	D	E	F	G	H		X	Y	Z
1												
2												
3												

3.2. Input and Output vectors Construction

Inputs:
The coding table for the system rules.

Output:
The input and output vectors for the system rules.
Updated coding table.
Highlights:
To make our concept clear in this part, assume we have the following rule
IF A THEN Z
and assume further that the number of conditions/inputs in the system rules are 5 (for example A,B,C,D and E). The above rule means, A should be true to conclude Z regardless of the values (true/false) of the other system inputs B,C,D and E. This simply means for example
IF A and not B THEN Z is still a true rule , and IF A and B and not C THEN Z is still a true rule unless a contradictory rule appears in the future.

The process:

a. Find number of the generated copies for each rule code (NC) based on the following formula

$$NC = n * \delta, \quad (3)$$

Where n is the number of inputs (for example 8 in Table 1) and δ is the copying factor. δ is a value that satisfies the condition $1 \leq \delta \leq 2^{n-z}$, where z is the number or remaining conditions other than those which have 1 or -1 in rule code. The larger is the value of δ, the more is the number of copies are generated. The minimum number of copies will be equal to n in case δ = 1.

b. Every input vector X_i is represented in the form $[i_1 \ i_2 \ i_3, ..i_n]$, where i_1 for example is the value assigned for the first input. Each copy of the rule code is identified by X_{ij} to indicate that it is the j^{th} copy of the i^{th} rule code.
c. For every rule code (row in the table) in the coding table do the following:
d. If the rule is having one condition c, NC number of copies will be randomly generated to have values 1 or -1 and indicated by 1' or -1' respectively for inputs other than c. The c value for all the copies will be always 1 for positive input condition and -1 for negative input condition. We use 1' and -1' to distinguish between these generated values and the original value of the condition c. This is important in the stage of rule extraction (rule code decoding). We do not use 0 as a don't care for other conditions other than c because 0 in Hopfield and BAMs indicates that 0 values means that the input/output will keep its previous value 1 or -1. So if we keep 0 in our computation as a don't care, we will not get correct results.
e. If the rule is having "and" relation between say i and j conditions, NC number of copies will be randomly generated to have values 1' or -1' for inputs other than i and j. The values for i and j will be always 1 or -1 for all the copies depending on their input value positive or negative.
f. If the rule is having "or" relation between say i and j conditions, two sets of copies will be generated, the first set will have the condition i value as 1 or -1 and the remaining conditions values will have either 1' or -1' values generated randomly. The second set will have the condition j value as 1 or -1 and the remaining conditions values will have either 1' or -1' values generated randomly. In this case we shall get 2*NC copies.
g. Output vector $Y_i = [o_1 \ o_2 \ o_3..o_m]$ is the output vector for the i^{th} rule code, where m is the number of outputs in the system. All the copies of the i^{th} rule code will have Y_i as output. This means the NC for a certain rule code will have the same output. Our assumption is that the output vector values are having exclusive or relation, which means only one of them can be true and the rest should be false, or one of them is false and the rest is true. The output vector will have 1 or -1 for the output to be achieved (1 for non-negated output and -1 for negated output). If we have a non-negated output, all the other output values should be -1', and if we have a negated output, all the other output values should be 1'.
It is to be noted that all the input copies are added to the code table.

3.3. Obtaining Weight Matrix

Input:
Code table -The input vectors extracted from the code table.

Output:
The weight matrix W
The process:

The proposed approach produces the weight matrix (W) that can be used to test and find the output of any given input. We follow the same approach used in BAMs in computing W. In the computation of W, every 1' is considered as 1 and every -1' is considered as -1. The following is the steps required to obtain the weight matrix.

a. For every rule code, we compute Rule Weight (RWi)

$$RW_i = \sum_{j=1}^{n} (Xij)T.(Yi) \quad (4)$$

b. Compute $W = \sum_{i=1}^{r} RWi$ (5)
Where r is the number of rule codes in the table.

3.4. Testing the System

Inputs:
The weight matrix W.
Code table.
The input (can be input or output) required to get its output/input.

Output:
The output of the given input, or the input for the given output.
The process:
The steps below are the steps to be followed to get output/input. This process includes testing and adapting the system:

a. Get the input and apply it to obtain the corresponding output using formula 2. In normal cases, if the given input vector is among the input vectors used to obtain the weight matrix, the exact expected output will be produced; this is normally obtained by following the BAMs approach. It is normal if the given input is not one of the inputs used to get W, the obtained output mostly will not be also included in the output list of the system.
b. If the obtained output is one of the outputs, then accept it, if it is not; try to find the closest output by computing the hamming distance. The hamming distance is the number of differences between the corresponding values in two vectors (this mechanism is used in Hopfield approach).
c. If the hamming distance d is less than or equal to threshold t, then consider the output with the most less hamming distance as the output for the given input. Otherwise, the output is considered to be the output of the given input known by the user. t can be computed as
   $t = m * c$ (6)

where m is the number of values in the output vector and c is acceptance coefficient which is $0 \leq c \leq 1$. When the user wants to be stricter, he/she can make the value of c less, and when he/she wants to be more flexible, he/she can increase the value of c. when c= 0, no tolerance is accepted and the same output should be considered. The worst case is when c=1, which means all output values in the output vector can be incorrect.

d. Since this is a testing stage, the output should be known by the user, the expected output is compared by the obtained output, if they are the same, the testing is over.
e. If the obtained output is not as the expected output, the input and the expected output are added to the code table and W is recomputed. This re-computation needs first determining only the inputs and output vectors as explained above for the recently added input and its corresponding output.

The above steps are applied if you start from input to output or if you start from output to input.

3.5. Running the System

Input:
The weight matrix W.
Code table
The input (can be input or output) required to get its output/input.
Output:
The input/output for the given input (can be input or output for the system) by the user
The process:
In this stage, steps from a to c in testing the system stage are typically performed. Then, if the obtained output is not among the output vectors listed in the code table, the given input and the corresponding obtained output are inserted into the code table, and W is recomputed.

3.6. Rule Extraction (Rule Code Decoding)

In this stage, we follow certain points to extract the rules, or what we call rule code decoding:

Input:
Rule code table
Output:
Extracted rules
The process:

a. For every code rule, we construct a rule with inputs as the conditions of the rule and the output as the action of the rule.
b. To construct the conditions, for every true input, use its name in the rule, if the input is false (-1 value); use its negated name in the rule. In this stage, we ignore 1' and -1' from our consideration.
c. For every rule code, if one or more input values have 1 or -1, the relation is checked in the table, if it is 0, "and" relation between the inputs is constructed, otherwise "or" relation between the input conditions is constructed.
d. To construct the action, for every true output (1), use its name in the action part of the rule. Similarly for negated output (-1), use its negated name in the action part of the rule. Again here 1' and -1' are ignored.
e. The rule should only be extracted once. Similar rules will be ignored.

4. Example

Let us assume we have the following rules
IF A THEN B
IF C THEN G
IF G and M THEN R
IF R or N THEN Z
IF X and ~S THEN E
The constructed table is the one shown in Table 2.

Table 2. The constructed table for the given set of rules. 0 relation means "and" relation, and 1 relation means "or" relation

	Inputs								Relation	Outputs				
#	A	C	G	M	R	N	X	S		B	G	R	Z	E
1	1									1				
2		1									1			
3			1	1					0 (and)			1		
4					1	1			1 (or)				1	
5							1	-1	0 (and)					1

Based on Table 2. W can be computed as below.

The input vector is to represent values for [A C G M R N X S] and the output vector represents [B G R Z E] in the same order.

For rule1 the value of A should be true (value 1) to conclude B. Based on this the output should be $Y_1 = [1 \quad -1' \quad -1' \quad -1' \quad -1']^T$. Where -1' is a value assigned by the algorithm to have exclusive or relation. This means B is true and all other outputs should be false. For the input, A should be true regardless of other input values.

Let us assume δ (copies factor) = 1.5, then NC = 8 x 1.5 = 12 copies, we generate all input values other than input A at random 1 or -1 and use 1' and -1' to represent them respectively, for example:

X_{11} = [1 -1' 1' -1' -1' 1' 1' -1']^T
X_{12}= [1 -1' -1' 1' 1' -1' -1' 1']^T
X_{13} = [1 1' 1' -1' -1' 1' 1' -1']^T and so on until we get 12 copies
All these input vectors have output Y_1
For the second rule C must be true to conclude G. This will give

X_{21} = [-1' 1 1' -1' -1' 1' -1' 1']^T
X_{22} = [1' 1 -1' -1' 1' -1' 1' 1']^T
X_{23} = [1' 1 -1' 1 ' 1' -1' 1' -1'] ^{T}

and so on until we get 12 copies and all of them have output
Y_2 = [-1' 1 -1' -1' -1']^T

Similarly for rule 3, to conclude R, G and M should be true. It has to be noted that the relation between G and M is "and" relation. This will give

X_{31} = [-1' -1' 1 1 -1' -1' 1' -1']^T
X_{32} = [-1' 1' 1 1 1' -1' -1' 1']^T
X_{33} = [1' -1' 1 1 -1' 1' -1' -1']^T
and so on until we get 12 copies and all of them have output
Y_3 = [-1' -1' 1 -1' -1']^T

For Rule 4, to conclude Z, R or N should be true. Because we have or relation, we create 24 copies, half of them ensures that R is true and the rest ensures that N is true.

X_{41} = [-1' -1' 1' -1' 1 1 -1' -1']^T
X_{42} = [-1' -1' -1' -1' 1 1 1' -1']^T
X_{43} = [-1' -1' -1' -1' 1 1 -1' -1']^T
…
$X_{413=}$ [-1 ' -1' -1' 1 ' 1 1 -1' -1']^T

The output for all X4 inputs is Y_4 =[-1' -1' -1' 1 -1']^T

For rule 5, the same process is done as that of rule 3, the only difference is that the values for X and S are always 1 and -1 consecutively. Below is an example related to X_{51}.

X_{51} = [-1' -1' -1' -1' -1' -1' 1 -1]

The output for all the copies for rule 5 (rule code 5) is
Y_5 = [-1' -1' -1' -1' 1]^T

Now based on the above, we include all the generated copies in the code table and compute the weight matrix W as below:
RW_1= $(X_{11})^T.Y_1$ + $(X_{12})^T.Y_1$+ $(X_{13})^T.Y_1$ +……$(X_{112})^T.Y_1$
RW_2= $(X_{21})^T.Y_1$ + $(X_{22})^T.Y_1$+ $(X_{23})^T.Y_1$ +……$(X_{212})^T.Y_2$

Similarly RW_3, RW_4 and RW_5 are computed. Now it is time to compute W
W = RW_1 + RW_2+RW_3 +RW_4+RW_5

5. Conclusion

In this paper a new algorithm to represent and adapt rules in rule based systems based on BAMs is proposed. The strength of BAMS is that it has a strong theoretical ground that can be used with full trust. The proposed algorithm utilizes BAMs to memorize and adapt rules in rule based systems and facilitates the deal with forward chaining and backward chaining as well. The proposed algorithm has advantages over other algorithms in terms of rule representation, inference mechanism and logic relations within a rule. The simplicity of rule representation through the code table made the rule extraction a simple process. Implementation of this algorithm is simple and rule adaptation process is still based on the theory of BAMs. Most of the previous algorithms deal with only one condition or with "and" relations between the conditions in the rule. In our algorithm, we also consider "or" relation between the conditions. The proposed algorithm discusses six issues, code table creation, input and output vectors construction, weight matrix calculation, testing the system, running the system and extracting the system rules. In code table creation, we construct a table where each rule is coded in terms of 1 and -1. In input and output vector, we generate copies of the rule codes and insert them in the code table, 1' and -1' are used to fill wildcard condition/outputs. '1 and -1' are used to be identified from the original 1's and -1's. In the weight matrix calculation, we follow the concepts used in BAMs. In the system testing stage, we follow a process to test the system learning performance and do necessary changes if needed. In the system running stage, the same process is done as testing stage, but the system tries to improve itself without the help of the user. In the rule extraction (rule code decoding) stage, we present how rules are extracted from the code table. Some of the future directions could be implementing the proposed algorithm with various rule based systems applied in various domains.

References

Bharadwaj, K., & Silva, J. (1998). Towards Integrating Hierarchical Censored Production Rule(HCPR) Based System and Neural Networks, in F. Oliveira (Ed.). LNAI (vol. 1515, pp.121-130), Springer.

Blount, J., Tauritz, D., & Mulder, S. (2011). Adaptive Rule-Based Malware Detection Employing Learning Classifier Systems: A Proof of Concept. 2011 IEEE 35th Annual Computer Software and Applications Conference Workshops, 110 – 115.

Cimino, M., Lazzerini, B., Marcelloni, F., & Ciaramella, A. (2012). An Adaptive Rule-Based Approach for Managing Situation-Awareness. Expert Systems with Applications 39(12) 10796–10811.

Farid, D., Mamun, M., Manderick, B., & Nowe, A. (2016). An Adaptive Rule-Based Classifier for Mining Big Biological Data", Expert Systems with Applications, vol. 64, 305-316.

Hewahi, N. (2002). A General Rule Structure (GRS), Journal of Information and Software Technology, vol. 44/8, 7-13.

Hewahi, N. (2004). Principles on Integrating General Rule Structure (GRS) Based Systems and Neural Networks, Proceedings of the International Conference on Artificial Intelligence, IC-AI '04, 174-180.

Hewahi, N. (2005). Training and Extracting Real Time system Symbolic Rules Using Neural Networks, Asian Journal of Information Technology, vol. 4, no. 10, 920-926.

Hewahi, N. (2011). Probabilistic Neural Networks for Rule Based Systems, International Journal of Advanced Research in Computer Science, vol.2, no. 2, 21-26.

Holland, J. (1985). Properties of the Bucket Brigade", Proceedings of the 1st International Conference on Genetic Algorithms, 1-7.

Hopfield, J. (1982). Neural Networks and Physical Systems with Emergent Collective Computational Abilities, Proceedings of the National Academy of Sciences of the USA, vol. 79, no. 8, 2554–2558.

Ioannou, P., Raouzaiou, A., Karpouzis, K., Pertselakis, M., Tsapatsoulis, N., & Kollias, S. (2004). "Adaptive Rule-Based Facial Expression Recognition", in Goebel, Randy, Tanaka, Yuzuru, Wahlster & Wolfgang (Eds.), LNAI, (vol. 3025, pp.466-475), Springer.

Jain, M., Dua, P., Dua, S., & Lukiw, W. (2013). Data Adaptive Rule-based Classification System for Alzheimer Classification, Journal of Computer Science & Systems Biology, vol. 6, 291-297.

Jithish, J., & Sankaran, S. (2017). A Hybrid Adaptive Rule Based System For Smart Home Energy Prediction, in CEUR Workshop Proceedings, vol. 1819.

Kosko, B. (1987). Bidirectional Associative Memories, IEEE Trans. Syst. Man Cybern. SMC-00, 000.

Okour, S., Maeder, A., & Basilakis, J.(2015). An Adaptive Rule-Based Approach to Classifying Activities of Daily Living, International Conference on Healthcare Informatics, 404 – 407.

Tabebordbar, A., & Beheshti, A. (2018). Adaptive Rule Monitoring System, IEEE/ACM 1st Inter. Workshop on Software Engineering for Cognitive Service (SE4COG), Gothenburg, Sweden, 45-51.

Towell, G., & Shavlik, J. (1991). Interpretation of Artificial Neural Networks: Mapping knowledge-Based Neural Networks into Rules, Advances in Neural Information Processing Systems, vol.4, 977-984.

Towell, G., & Shavlik, J. (1994). Knowledge based artificial neural networks, Artificial Intelligence, 70, 119-165.

Wang, O. (2017). Adaptive Rules Model: Statistical Learning for Rule-Based Systems, Machine Learning [cs.LG], Université Paris-Saclay, PhD Thesis.

Nabil M. Hewahi obtained his PhD degree in Computer Science from Jawaharlal Nehru University, New Delhi, India in 1994, and M.Tech degree in Computer Science and Engineering from Indian Institute of Technology, Bombay, India in 1991. He is now with the university of Bahrain, Bahrain and the Islamic University of Gaza, Palestine. Dr. Hewahi is a full professor of Computer Science (Artificial Intelligence) since 2006. He published over 65 papers in well-known journals and conferences. His main research interest is Intelligent Systems including machine learning and knowledge representation.