Analysis of the Algorithm for the New Architecture of SFARTMAP based on the Extended Definition of Complementation of Fuzzy Set

Pranamika Kakati
Department of Computer Science,
Gauhati University, Guwahati, Assam, India

ABSTRACT
This article analyses the algorithm for the new architecture of SFARTMAP based on the extended definition of complementation of Fuzzy set. The extended definition of complementation of Fuzzy set is based on the fact that Fuzzy membership function and Fuzzy membership value for the complement of a Fuzzy set are two different things. In this paper, the analysis of the algorithm is carried out using Big O notation to find the order of growth for it.

General Terms
Algorithm for Simplified Fuzzy ARTMAP, Time Complexity

Keywords
Complement of a Fuzzy set, Fuzzy membership function, Fuzzy membership value, Fuzzy reference function, Fuzzy set, Simplified Fuzzy ARTMAP.

1. INTRODUCTION
The most common metric for calculating time complexity is Big O notation. Big O gives the upper bound for time complexity of an algorithm. The algorithm for the new architecture of SFARTMAP is based on the extended definition of Fuzzy set introduced by Baruah[2,3,4]. According to Baruah , to define a Fuzzy set two functions namely- Fuzzy membership function and –Fuzzy reference function are necessary. Fuzzy membership value is the difference between Fuzzy membership function and Fuzzy reference function. Fuzzy membership function and Fuzzy membership value are two different things. Also Neog and Sut [4] have generalized the concept of complement of a Fuzzy set, introduced by Baruah[2,3], when the Fuzzy reference function is not zero and defined arbitrary Fuzzy union and intersection extending the definition of Fuzzy sets given by Baruah [2, 3]. On the basis of this extended definition of complementation[2,3,4] of Fuzzy set based on reference function we have established a new architecture of SFARTMAP[14] and redesigned the algorithm for it together with the demonstration of its application on some example data. In this article, our aim is to find the order of growth for this algorithm using big O notation.

The overall organization of this paper is as follows. Section 2 overviews the new architecture of SFARTMAP based on the extended definition of complementation. In section 3 we discuss the algorithm for the new architecture of SFARTMAP. We analyze the time complexity for the algorithm of the new SFARTMAP in section 4. Finally, some conclusions are given in section 5.

2. THE NEW SFARTMAP ARCHITECTURE
Simplified Fuzzy ARTMAP (SFARTMAP) proposed by Tom Kasuba [9] in 1993 is a vast simplification of Carpenter and Grossberg's Fuzzy ARTMAP (in 1992) with reduced computational overhead and architectural redundancy. SFARTMAP is basically a two layer Neural Network containing- an input and an output layer. We can view the architecture of SFATRMAP in Figure 1.

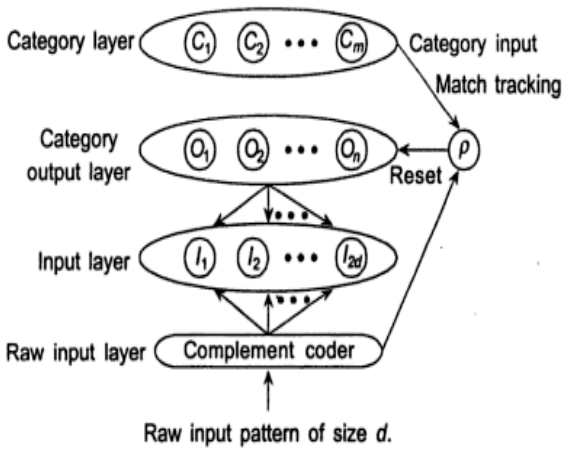

Figure 1 Simplified Fuzzy ARTMAP architecture

In the SFARTMAP, the input to the network enters through the complement coder where the input string is stretched by adding its complement also. Then the complement coded input enters into the input layer and remains there. Weights (W) from each of the output category nodes flow down to the input layer. The category layer only contains the names of the M number of categories that the network has to learn. Vigilance parameter and match tracking are two techniques of the network architecture which are mainly in use for network training.

The vigilance parameter, ρ can lie in the range from 0 to 1.ρ is user selectable which controls the granularity of the output node encoding. Thus, while high vigilance values makes the output node much fussier during pattern encoding, low vigilance causes the output node to be liberal during the encoding of patterns.

The match tracking technique of the network adjusts the vigilance values, which is responsible for network learning.

Therefore, when an error occurs in the training phase during the classification of patterns, i.e. when the selected output node does not match the same output category corresponding to the input pattern presented, match tracking is used. Depending on the situation, match tracking may result in the network adjusting its learning parameters or the network opening new output nodes.

2.1 Input Normalization

Complement coding is used for input normalization which represents the presence of a particular feature in the input pattern and its absence. For example,

If the input vector , a={(0.7,0.3)} [we have taken 0.7 > 0.3 because we consider that every Fuzzy number is defined with a membership function value (μ_m)and a reference function value(μ_r) with the condition $0 \leq \mu_r \leq \mu_m \leq 1$ as (μ_m , μ_r)] is a single dimension ordered pair vector then according to Baruah's definition of Fuzzy set, under the 3 possible cases[14], we can compute the complement of vector a as follows:

a^c = {(0.3,0.0),(1.0,0.7)}. Therefore, the complement coded input vector obtained by concatenating a^c with a is given by the vector,

AI = {a, a^c} = {(0.7, 0.3), (0.3, 0.0), (1.0, 0.7)} which is known as Augmented Input vector.

Similarly we can obtain complement vector a^c for a n-dimensional ordered pair input vector a given by,

a = {(0.2, 0.1), (0.4,0.3),(0.1,0.0)}as
a^c={(0.1,0.0),(1.0,0.2),(0.3,0.0),(1.0,0.4), (0.0,0.0),(1.0,0.1)}

and the Augmented Input vector as

AI=

{(0.2,0.1),(0.1,0.0),(1.0,0.2),(0.4,0.3),(0.3,0.0),(1.0,0.4), (0.1,0.0),(0.0,0.0),(1.0,0.1)}

The normalization process is essential since SFARTMAP needs all its input values to lie between 0 to 1.

The learning equations of the architecture call for the computation of |***I***|. Here, '| |' is the norm of a vector defined as

$|p| = \sum_{i=1}^{n}(p_i)$,for p=(p_1, p_2, ……., p_n).

For Example, for a n-dimensional ordered pair vector a= {(0.2, 0.1), (0.4,0.3), (0.1,0.0)}, |a| is given by

|a|= |{(0.2, 0.1),(0.4,0.3),(0.1,0.0)}|

=|(0.2+0.4+0.1) – (0.1+0.3+0.0) |

=|0.7-0.4|=0.3

2.2 Output Node Activation

Network Training Phase. When the complement coded forms of input patterns are introduced to the SFARTMAP, all output nodes become active to varying degrees.

Activation Function

The output activation, denoted by T_j and referred to as the activation function for the jth output node, where W_j is the corresponding top-down weight, is given by

$$T_j(I) = \frac{|I \wedge W_j|}{\alpha + |W_j|}$$

where, α is kept as small as close to 0 usually about 0.0000001. We compute $I \wedge W_j$ using the intersection operation of Fuzzy set as discussed in [14]. For Example,

if I= {(0.4,0.1),(0.5,0.2),(0.9,0.5)}

and W_j= {(0.3,0.0),(0.6,0.4),(1.0,0.1)} then

$I \wedge W_j$=
{(min(0.4,0.3),max(0.1,0.0)),(min(0.5,0.6),max(0.2,0.4)),(min (0.9,1.0),max(0.5,0.1)) }

={(0.3,0.1),(0.5,0.4),(0.9,0.5)}

Now regarding the activation function value, there may be two possible cases:

Case 1: Single Winner
In the event of only one node emerging as the winner, the node which records the highest activation function is considered winner .i.e.

Winner = max (T_j)

Case 2: More than one winner
In the event of more than one node emerging as the winner due to the same activation function value, some mechanism: such as choosing a node with the smallest index : may be designed to break the tie.

The category corresponding the winner is the one to which the given input pattern belongs to, as classified by the network with the help of match function.

Match Function
The match function which helps to decide whether the network must adjust its learning parameters is given by

$$\frac{|I \wedge W_j|}{|I|}$$

As stated earlier, the match function in association with the vigilance parameter decides on whether a particular output node is good enough to encode a given input pattern or whether a new output node should be opened to encode the same. Depending on these two situations two possible cases may arise:

Case 1: Resonance State
If the match function value is greater than the vigilance parameter value then the network is said to be in a state of resonance. For a node to show resonance, it is important that it not only encodes the given input pattern but should also represent the same category as that of the input pattern. Depending on this condition two possible cases for an output node may occur:

Case I: Output node represents the same category as that of the input pattern
In this case, the match function value is greater than the vigilance parameter value and also the output node represents the same category as that of the input pattern. Hence the particular output node is fit enough to learn the given input pattern and thus can update its weights by the weight update equation. Thus the output node is said to be in the state of resonance.

Case II: Output node does not represent the same category as that of the input pattern

Here, also the match function value is greater than the vigilance parameter value but the selected output node does not match the same output category corresponding to the input pattern. So the particular output node is not in the state of resonance and is not fit enough to learn the given input pattern and hence **match tracking** is used here. In the match tracking mechanism, the vigilance parameter ρ value is incremented by a small value and after that it is checked to find whether some more top down weight nodes exist or not .If some more top down weight nodes exist then the next highest winner node weight among the top down weight nodes is considered and again corresponding to this weight node, the value of match function is calculated and compared with the new value of the vigilance parameter to find a suitable output node in resonance state. If still an output node is not found fit to learn the given input pattern then a new top-down weight corresponding to the input pattern is created and linked to the output category of the given input pattern.

Case 2: Mismatch Reset

On the other hand, if the vigilance parameter value is greater than match function value then the network is said to be in a state of mismatch reset .Such a state only means that the particular output node is not fit enough to learn the given input pattern and thus cannot update its weights even though the category of the output node may be the same as that of the input pattern. This is so, because the output node has fallen short of the expected encoding granularity indicated by the vigilance parameter. For such case, we find to know whether some more top down weight nodes exist. If some more top down weight nodes exist then the next highest winner node weight node among the top down weight nodes is considered and again the value of match function corresponding to this weight node is calculated and compared with the vigilance parameter value to find a suitable output node. If still an output node is not found fit to learn the given input pattern then a new top-down weight corresponding to the input pattern is created and linked to the output category of the given input pattern.

The weight updating equation of an output node j when it goes to learn the given input pattern I is given by

$$W_j^{new} = \beta\,(\,I \wedge W_j^{old}) + (1-\beta)W_j^{old}$$

where $0 < \beta \leq 1$

Network Inference Phase. Once the network has been trained, the inference of patterns, known or unknown, i.e. the categories to which the patterns belong, may be easily calculated. This is achieved by passing the input pattern into the complement coder and then to the input layer. All the output nodes calculate the activation functions with respect to the input. The winner, i.e. the node with the highest activation function, is chosen. Finally, the category to which the winning output node belongs is the one to which the given input pattern is classified by the network.

3. ALGORITHM OF THE NEW SFARTMAP ARCHITECTURE

We propose the algorithms for Training and Inference phases of SFARTMAP on the basis of extended definition of complementation as follows:

SFARTMAP –Training phase

Step 1: Choose an appropriate value for the vigilance parameter ($0 < \rho < 1$) and a small value for . Set NO_OF_TRAINING_EPOCHS to the desired number of training epochs and COUNT_OF_TRAINING_EPOCHS to 0.

Step 2: i ← 1;

COUNT_OF_TRAINING_EPOCHS= COUNT_OF_TRAINING_EPOCHS+1;
While (COUNT_OF_TRAINING_EPOCHS ≤ NO_OF_TRAINIG_EPOCHS)
Repeat Steps 3 - 12;

Step 3: Input the pattern vector $I_i = (a_{i1}, a_{i2}, a_{i3},\ldots\ldots, a_{id})$ of dimension d and its category C_i.

Step 4: Compute the augmented input vector using the extended definition of complementation of Fuzzy set under the 3 possible cases[14].

Step 5: If AI_i is the first input in the given category C_i set the top down weight vector W_i as AI_i
i.e. $W_i = AI_i$;
Link W_i to the category C_i.
Go to step 12.

Step 6: If AI_i is an input pattern vector whose category already exits then compute the activation function $T_j(AI_i)$ for each of the existing top-down weight nodes W_j

$$T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|};$$

Step 7: Choose that top-down weight node k which records the highest activation function
$T_k(AI_i) = \max_j T_j(AI_i)$

Step 8: Compute the match function $MF_k(AI_i)$ of the winning node k;
If $MF_k(AI_i) > \rho$ and C_i is same as that category C_k linked to W_k
Then update weight vector W_k as $W_k^{new} = (1-\beta)W_k^{old} + \beta(I \wedge W_k^{old})$. (Here choosing β=1)
Go to step 12.

Step 9: If $MF_k(AI_i) > \rho$ and C_i is not the category C_k linked to W_k then
Undertake match tracking by setting ρ to $MF_k(AI_i)$ and incrementing by a small value ε.
$\rho = MF_k(AI_i) + \varepsilon$;
If some more top down weight nodes exist
Then consider the next highest winner W_k among the top-down weight nodes;
Go to step 8 ;
else go to step 11;

Step 10: If $MF_k(AI_i) < \rho$
then
If some more top down weight nodes exist
then

Consider the next highest winner W_k among the top-down weight nodes.
Go to step 8;
Else go to step 11;

Step 11: Create a new top-down weight node W_1 such that $W_1=AI_i$ and link the node to the category C_i;

Step 12: If no more input patterns then go to step 13;
else
$i \leftarrow i+1$
goto Step 3;

Step 13: goto step 2;

SFARTMAP –Inference phase

Step 1: Let W_j , j=1,2,……..,s indicate s top-down weight vectors obtained after training the network with a given set of training patterns;

Let I_i be the inference pattern set each of whose category is to be inferred by the network;

$i \leftarrow 1$;

Step 2: Read input I_i;

Step 3: Compute the augmented input AI_i;

Step 4: for $j \leftarrow 1$ to s

Compute the activation functions

$$T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|};$$

Step 5: Choose the winner k among the S activation functions

$$T_k(AI_i) = \max_j T_j(AI_i)$$

Step 6: Output category C_k linked to $T_k(AI_i)$ as the one to which I_i belongs to.

Step 7: If no more inference pattern vectors
then exit
else $i \leftarrow i+1$;
go to step 2.

4. ANALYSIS OF THE PROPOSED ALGORITHM

Now we analyze the proposed algorithm for SFARTMAP architecture using Big O- notation. The most common metric for calculating time complexity is Big O notation. Big O gives the upper bound for time complexity of an algorithm.

SFARTMAP –Training phase

Step 1: Choose an appropriate value for the vigilance parameter ($0 < \rho < 1$) and a small value for . Set NO_OF_TRAINING_EPOCHS to the desired number of training epochs and COUNT_OF_TRAINING_EPOCHS to 0.

Step 1 executes 3 assignment statements:
It assigns values for the learning parameters ρ and α
It sets NO_OF_TRAINING _EPOCHS to the desired no of training epochs and
It initializes COUNT_OF_TRAINING_EPOCHS to 0.

Therefore step 1 is a sequence of simple statements, each of which executes for constant amount of time regardless of input size. Hence the runtime for each statement is denoted by O(1) and therefore the total runtime for step 1 is O(1) + O(1) + O(1) = O(1).

Step 2: $i \leftarrow 1$;

COUNT_OF_TRAINING_EPOCHS= COUNT_OF_TRAINING_EPOCHS+1;
While (COUNT_OF_TRAINING_EPOCHS ≤ NO_OF_TRAINIG_EPOCHS)
Repeat Steps 3 - 12;

Step 2 contains an assignment statement to initialize the variable i to 1 and a while loop that controls the whole algorithm from step 3 to 12 containing an assignment statement to linearly increment the counter variable COUNT_OF_TRAINING_EPOCHS. The execution of the first assignment statement is independent of input and that of the second assignment statement with the while loop depends on the input size i.e. the no. of training epochs (suppose, n). Therefore the first assignment statement executes for a constant amount of time and the second assignment statement with the loop executes for n times. Hence the run time of the first assignment statement is denoted by O(1) and that of the loop with the second assignment statement is denoted by O(n).Thus the overall run time for the step 2 is O(1) + O(n) = O(n).

Step 3: Input the pattern vector $I_i = (a_{i1}, a_{i2}, a_{i3},……., a_{id})$ of dimension d and its category C_i.

Step 3 reads an input vector I_i of dimension d in the given category C_i. Therefore this is an input statement and it executes for constant amount of time. So its run time is O(1). Since this input statement lies inside the while loop in step 2 and the loop executes for n times, as a result the input statement also executes for n times. Hence the runtime for step 3 is n * O(1) = O(n).

Step 4: Compute the augmented input vector using the extended definition of complementation of Fuzzy set under the 3 possible cases[14].

Step 4 computes the augmented input vector AI_i for the given input pattern vector I_i using the extended definition of fuzzy set under any one of the 3 possible cases. Therefore it involves if statements which executes for constant time and hence its runtime is O(1). Since the statements in step 4 lie within the while loop in step 2 and the loop executes for n times, therefore the runtime for step 4 is n * O(1) = O(n).

Step 5: If AI_i is the first input in the given category C_i set the top down weight vector W_i as AI_i
i.e. $W_i = AI_i$;
Link W_i to the category C_i.
Go to step 12.

Step 5 has an if statement to decide whether the given input vector I_i is the first input seen by the network in the given category C_i. If so it sets the corresponding top-down weight vector W_i as AI_i and then link W_i to the category C_i. After that it goes to read the next input vector. Therefore step 5 contains

an if statement and a go to statement, each of which executes for constant amount of time. As the sequence of statements of step 5 are contained within the while loop of step 2 and since the loop executes for n times, therefore the total runtime for step 5 is n * O(1) + n * O(1) = O(n) + O(n) = O(n).

Step 6: If AI_i is an input pattern vector whose category already exits then compute the activation function $T_j(AI_i)$ for each of the existing top-down weight nodes W_j

$$T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|};$$

Step 6 also has an if statement which determines whether the given input I_i is an input pattern whose category already exists and if it is so then it computes the activation function $T_j(AI_i)$ w.r.t. the given input pattern I_i for each of the existing top down weight nodes W_j using loop having the computational statement $T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|}$ which involves basic operations independent of input size and hence executes for constant amount of time . Therefore this loop containing the computational statement lies inside the while loop in step 2 and it executes m times (suppose, total no. of existing top-down weight nodes W_j is m) for every iteration of the loop in step 2 and thus for a total of m x n times i.e. n^2 times in general. Hence step 6 includes an if statement having a loop which executes for n^2 times. So runtime for step 6 is n * O(n) = $O(n^2)$.

Step 7: Choose that top-down weight node k which records the highest activation function

$$T_k(AI_i) = \max_j T_j(AI_i)$$

Step 7 finds out the winner node choosing the top-down weight node k which records the highest activation function $T_k(AI_i)$ among the already calculated j activation function values in step 6, using a loop. So this loop lies inside the outer loop in step 2 and hence it executes m times (suppose, m is the total no. of calculated activation function values) for each iteration of the outer loop in step 2. Therefore step 7 executes for a total no. of m x n i.e. n^2 times in general and so its run time is n * O(n) = $O(n^2)$.

Step 8: Compute the match function $MF_k(AI_i)$ of the winning node k;
If $MF_k(AI_i) > \rho$ and C_i is same as that category C_k linked to W_k
Then update weight vector W_k as $W_k^{new} = (1-\beta)W_k^{old} + (I \wedge W_k^{old})$. (Here choosing β=1)
Go to step 12.

Step 8 computes the match function $MF_k(AI_i)$ of the winning node k. After that it uses an if statement to check whether the match function value $MF_k(AI_i)$ corresponding to the winner node k is greater than the vigilance parameter value ρ and the given input category C_i is same as that category C_k of the winner node linked to the corresponding top-down weight node W_k. If the if-condition becomes true then the selected winner node k is considered fit enough to encode the given input pattern I_i and hence it can update its weight vector W_k using the weight update equation and after that it goes to read the next input vector. Therefore step 8 contains a computational statement, an if-statement and a goto statement: each of which executes for constant amount of time. Since the statements in step 8 execute within the while loop in step 2. Therefore the total runtime for step 8 is n*O(1) + n*O(1) + n*O(1) = O(n) + O(n) + O(n)= O(n).

Step 9: If $MF_k(AI_i) > \rho$ and C_i is not the category C_k linked to W_k then
Undertake match tracking by setting ρ to $MF_k(AI_i)$ and incrementing by a small value ε.
$\rho = MF_k(AI_i) + \varepsilon$;
If some more top down weight nodes exist
Then consider the next highest winner W_k among the top-down weight nodes;
Go to step 8 ;
else go to step 11;

Step 9 also contains an if statement to check whether the match function value $MF_k(AI_i)$ corresponding to the winner node k is greater than the vigilance parameter value ρ and the given input category C_i is not same as that category C_k of the winner node linked to the corresponding top-down weight node W_k. If the if-condition becomes true then it undertakes match tracking mechanism by setting ρ value to $MF_k(AI_i)$ and incrementing it by a small value $\in$. After that it uses another if statement and finds to know whether some more top-down weight nodes exist to consider the next highest winner node W_k among the top-down weight nodes using a loop and go to repeat step 8 to find a suitable node in resonance state to encode the given input pattern I_i. If still a node is not found fit enough to encode the given input pattern then it goes to execute step 11. Therefore step 9 involves an if-statement having an assignment statement, an if-statement having a loop and a go to statement. The if-statements and go to statement execute for constant amount of time and the loop executes for n times. As the sequence of statements in step 9 lie inside the while loop in step 2 and the loop in step 2 executes for n times, hence the overall run time for step 9 is n * O(1) + n * O(n) + n * O(1) = O(n) + $O(n^2)$ + O(n) = $O(n^2)$.

Step 10: If $MF_k(AI_i) < \rho$
then
If some more top down weight nodes exist
then
Consider the next highest winner W_k among the top-down weight nodes.
Go to step 8;
Else go to step 11;

Step 10 also has an if statement to check whether the match function value $MF_k(AI_i)$ corresponding to the winner node k is less than the vigilance parameter value ρ and if the if-condition becomes true it uses another if-statement to know whether some more top-down weight nodes exist to consider the next highest winner node W_k among the top-down weight nodes using a loop and go to repeat step 8 to find a suitable node in resonance state to encode the given input pattern I_i. If still a node is not found fit enough to encode the given input pattern then it goes to execute step 11.Therefore step 10 involves a nested if-statement having a loop and a go to statement. The if-statements and go to statement execute for constant amount of time and the loop executes for n times. Since the sequence of statements in step 10 are contained in the while loop in step 2, so the total run time for step 10 is n * O(1) + n * O(n) + n * O(1) = O(n) + $O(n^2)$ + O(n) = $O(n^2)$.

Step 11: Create a new top-down weight node W_1 such that W_1=AI_i and link the node to the category C_i;

Step 11 creates a new top-down weight node W_i corresponding to the given input pattern I_i and assign the augmented input values AI_i to it and link it to the category C_i of the given input pattern. Therefore step 11 has two assignment statements, each of which executes for constant time and as the statements execute inside the loop statement in step 2, hence the total run time for step 11 is n * O(1) + n * O(1) = O(n) + O(n) = O(n).

Step 12: If no more input patterns then go to step 13;
else
$i \leftarrow i + 1$

goto Step 3;

Step 12 uses an if-else statement to find to know whether there are more input patterns or not. If there are no more input patterns then it goes to step 13 to go to step 2 to exit the algorithm else it increments the variable i linearly and then goes to step 3 to repeat the whole algorithm again. Therefore step 12 has an if-else statement containing an assignment statement and a go to statement: each of which executes for constant amount of time inside the while loop in step 2. Hence the total run time for step 12 is n * O(1) = O(n).

Step 13: goto step 2;
Step 13 has a single go to statement which executes for constant amount of time and so the run time for step 13 is O(1).

Therefore summarizing the run times of every steps, the overall run time for the entire algorithm is
$O(1) + O(n) + O(n) + O(n) + O(n) + O(n^2) + O(n^2) + O(n) + O(n^2) + O(n^2) + O(n) + O(n) + O(1) = O(n^2)$.

SFARTMAP –Inference phase

Step 1: Let W_j , j=1,2,……..,s indicate s top-down weight vectors obtained after training the network with a given set of training patterns;

Let I_i be the inference pattern set each of whose category is to be inferred by the network;

$i \leftarrow 1$;

Step 1 declares s top-down weight vectors W_j , j=1,2,3,…….., s obtained after training the network and the set of inference patterns I_i. Also it contains an assignment statement to initialize the index variable i of the loop statement that controls the algorithm from step 2 to step 7.The two declaration statements do not execute and the assignment statement executes for constant amount of time. Therefore the total run time for step 1 is O(1).

Step 2: Read input I_i;
Step 2 reads an input pattern I_i. Therefore this is an input statement and it executes for constant amount of time. Since this statement executes in the loop that controls step 2 to 7 and depends on input size i.e. the no. of inference patterns, hence the total run time for step 2 is n * O(1) = O(n).

Step 3: Compute the augmented input AI_i;

Step 3 computes the augmented input values AI_i for the given input pattern vector I_i using the extended definition of fuzzy set under any one of the 3 possible cases. Therefore step 3 involves if statements having computational statement including basic operations and hence executes for a constant amount of time. As the statement in step 3 executes in loop that depends on input size, so its run time is n * O(1) = O(n).

Step 4: for $j \leftarrow 1$ to s

Compute the activation functions

$$T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|};$$

Step 4 has a loop to compute activation function values corresponding to the given input pattern I_i for each of the existing top-down weight nodes W_j using the computational statement $T_j(AI_i) = \frac{|AI_i \wedge W_j|}{\alpha + |W_j|}$ which involves basic operations independent of input size and hence executes for constant amount of time. Therefore the execution of the loop containing this computational statement depends on the input size i.e. the no. of existing top-down weight nodes (suppose, m). Hence the loop executes m times for every iteration of the outer loop containing step 2 to step 7 i.e. for a total no. of m x n times i.e. n^2 times in general. Therefore total run time for step 4 is n * O(n) = $O(n^2)$.

Step 5: Choose the winner k among the S activation functions

$$T_k(AI_i) = \max_j T_j(AI_i)$$

Step 5 finds out the winner node choosing the top-down weight node k which records the highest activation function value $T_k(AI_i)$ among the already calculated j activation function values in step 4, using a loop. Thus the loop executes depending on the no. of calculated activation function values (suppose, m). As the loop executes inside the outer loop controlling step 2 to 7, hence it executes a total of m x n times i.e. n^2 times in general. Therefore total run time for the step 5 is n * O(n) = $O(n^2)$.

Step 6: Output category C_k linked to $T_k(AI_i)$ as the one to which I_i belongs to.

Step 6 outputs the category C_k of the winner node as the one to which the given input pattern belongs to as classified by the network. Therefore step 6 is an output statement that executes for constant amount of time. Since step 6 lie inside the loop statement including step 2 to 7, so it executes for n times. Thus the total run time for the step 6 is n * O(1) = O(n).

Step 7: If no more inference pattern vectors
then exit
else $i \leftarrow i + 1$;
go to step 2.

Step 7 uses an if-else statement to check whether some more inference patterns exist. If there are no more inference pattern vectors then it exits the algorithm else it increments the index variable i of the loop containing step 2 to 7 linearly and then go to step 2 to repeat the same procedure again. The if-else statement, exit statement, assignment statement and go to statement : each executes for constant amount of time inside the loop iterating step 2 to 7, therefore the run time for step 7 is n * O(1) = O(n).

Therefore summarizing the run times of every steps, the overall run time for the entire algorithm is
$O(1) + O(n) + O(n) + O(n^2) + O(n^2) + O(n) + O(n) = O(n^2)$.

5. CONCLUSION

In this article, we have focused on the new architecture of SFARTMAP based on the extended definition of complementation using reference function where it is believed that Fuzzy membership function and Fuzzy membership value for the complement of a Fuzzy set are two different things. Next, we have discussed the algorithm for the new architecture of SFARTMAP. Also we have analyzed the time complexity of the algorithm. Finally, it has been found that the overall runtime for the algorithm is $O(n^2)$.

6. ACKNOWLEDGMENT

The author would like to thank Hemanta K. Baruah, Vice Chancellor, Bodoland University, Kokrajhar, Assam for his valuable suggestions and guidance, in preparing this article.

7. REFERENCES

[1] L.A.Zadeh, " *Fuzzy Sets*" , Information and Control, 8, pp. 338-353, 1965.

[2] Hemanta K. Baruah, "*Towards Forming A Field Of Fuzzy Sets*" International Journal of Energy, Information and Communications, Vol.2, Issue 1, pp. 16-20, February 2011.

[3] Hemanta K. Baruah, "*The Theory of Fuzzy Sets: Beliefs and Realities*", International Journal of Energy, Information and Communications, Vol. 2, Issue 2,pp. 1-22, May,2011.

[4] Tridiv Jyoti Neog , Dusmanta Kumar Sut, "*Theory of Fuzzy Soft Sets from a New Perspective*", International Journal of Latest Trends in Computing, Vol. 2, No 3, September,2011.

[5] Eulalia Szmidt, Janusz Kacprzyk, "*Medical Diagnostic Reasoning Using a Similarity Measure for Intuitionistic Fuzzy Sets*" Eighth Int. Conf. on IFSs, Varna, 20-21 June 2004, NIFS Vol. 10 (2004), 4, 61-69.

[6] Hong-mei Ju , Feng-Ying Wang , "*A Similarity Measure for Interval –valued Fuzzy Sets and Its Application in Supporting Medical Diagnostic Reasoning*", The Tenth International Symposium on Operation Research and Its Applications(ISORA 2011), Dunhuang, China, August 28-31, 2011.

[7] Szmidt E., Kacprzyk J. (2001), *"Entropy for intuitionistic Fuzzy sets".* Fuzzy Sets and Systems, vol. 118, No. 3, pp. 467-477.

[8] Hongmei Ju (2008). *"Entropy for Interval-valued Fuzzy Sets"*,Fuzzy information and engineering, Volume 1,358-366.

[9] Kasuba , Tom(1993), Simplified Fuzzy ARTMAP, *AI Expert*, November, pp.18-25.

[10] Pranamika Kakati (2013). "*A Study on Similarity Measure for Fuzzy Sets*" International Journal of Advanced Research in Computer Science and Softaware Engineering, vol. 3 Issue 8, pp.97-103, August 2013.

[11] Pranamika Kakati (2013) *"A New Similarity Measure for Fuzzy Sets with the Extended Definition of Complementation"*International Journal of Soft Computing and Engineering, Volume 3 Issue 4, pp.203-207, September, 2013.

[12] Pranamika Kakati (2013) "*A Note on the New Similarity Measure for Fuzzy sets*" International Journal of Computer Applications Technology and Research, Volume 2 Issue 5, pp.601-605, September-October, 2013.

[13] Pranamika Kakati (2013) "*The New Similarity Measure for Fuzzy Sets and Its application to Medical Diagnostic Reasoning*" International Journal of Computer Application, Volume 80- No 15, pp. 13-17, October 2013.

[14] Pranamika Kakati (2013) "*A New Architecture of Simplified Fuzzy ARTMAP with the Extended Definition of Complementation*" International Journal of Computer Technology and Applications, Volume 4 Issue 5, pp. 772-784, September-October, 2013.

[15] Pranamika Kakati, Hemanta K. Baruah (2013) "*The New Architecture of Simplified Fuzzy ARTMAP for Supporting Medical Diagnostic Reasoning"* International Journal of Computer Application, Volume 81- No 17, pp. 16-19, November 2013.