2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Universal Approximation Propriety of Flexible Beta Basis Function Neural Tree

S. Bouaziz, Adel M. Alimi, Ajith Abraham

***Abstract*— In this paper, the universal approximation propriety is proved for the Flexible Beta Basis Function Neural Tree (FBBFNT) model. This model is a tree-encoding method for designing Beta basis function neural network. The performance of FBBFNT is evaluated for benchmark problems drawn from time series approximation area and is compared with other methods in the literature.**

I. Introduction

THE initiative of using Beta function for designing Artificial Neural Network was introduced by Alimi in 1997 [1] and in this case the network is called Beta Basis Function Neural Network (BBFNN). The BBFNN is a three layer feed-forward neural network that generally adopts a linear transfer function for the output layer and a Beta function as a nonlinear transfer function for the hidden units.

The Beta function has several advantages over the Gaussian function, such as its ability to generate more rich shapes (asymmetry, linearity, etc.) [2] and its great flexibility. In addition, Alimi *et al.* [3] and Aouiti *et al.* [4] demonstrated that BBFNN can be considered as a universal approximator. Therefore several success researches have been achieved in the use of the BBFNN for classification (pattern recognition) [5], [6], prediction [7], [8], etc.

Although conventional representation of BBFNN has a number of advantages such as better approximation capabilities and simple network topologies, adapting the matrix-encoding method suffers from slow premature convergence characteristics and makes the BBFNN's structure difficult to regulate. These reasons encourage us to use the tree-based encoding method for representing a BBF neural network. The new representation called Flexible Beta Basis Function Neural Tree (FBBFNT) [9], [10], [11], [12], is more flexible than the classical BBFNN seen that it can find automatically the number of nodes as well as the number of hidden layers. The FBBFNT is evolved by a hybrid algorithm with two levels: structure evolution and parameter evolution using evolutionary computation [13], [14], [15]. The performance of the evolving FBBFNT is tested for approximating some nonlinear systems.

In fact, the problem of function approximation can be considered as an aspect of neural networks learning. In the case that the neural network task is a task of modeling a physical process, it is reasonable to assume that the measured outputs of the process obey deterministic laws and to seek a mathematical expression of the function to be approximated [16]. The universal approximation property is so a necessary property of the model used for this purpose but it is not sufficient. In practice, the functions are defined by determining a finite set of couples (Input-Output) which not determine these functions univocally, the goal of learning is to find the most parsimonious solution. Therefore we are motivated in the current paper to justify that FBBFNT is also a universal approximator.

The remainder of this paper is planned as follows: Section II describes the Beta function and its proprieties. Section III introduces the Flexible Beta Basis Function Neural Tree design procedure and mathematical model. The universal approximation propriety of the FBBFNT network will be proved in Section IV. The set of some simulation results for time series approximation are provided in Section V. Finally, some concluding remarks are presented in Section VI.

S. Bouaziz and Adel M. Alimi are with REsearch Groups in Intelligent Machines (REGIM), University of Sfax, National School of Engineers (ENIS), BP 1173, Sfax 3038, Tunisia, (email: {souhir.bouaziz, adel.alimi}@ieee.org) Ajith Abraham is with 1Machine Intelligence Research Labs, WA, USA; 2IT4Innovations, VSB-Technical University of Ostrava, Czech Republic, (email: ajith.abraham@ieee.org).

II. Beta Function

The Beta function is the name used by Legendre and Whittaker and Watson (1990) for the Beta integral (also called the Eulerian integral of the first kind).

The first time where the Beta function was used as transfer function for neural networks was by Alimi [1].
This function was chosen as a transfer function for many reasons [3], [17], [18], including, its large flexibility (Figure 1) and its ability to generate rich shapes (asymmetry, linearity, etc.) [2].

A. Beta Function: definitions and proprieties

In the one-dimensional case, and if x_0 and x_1 are in $\mathbb{R}$ such that $x_0 < x_1$ then the Beta function is defined by $\beta(x) = \beta(x; x_0, x_1, p, q)$ with four possible cases.

- **Case 1:** $p > 0,\ q > 0$

$$\beta(x) = \beta(x; x_0, x_1, p, q) = \begin{cases} \left[\frac{x-x_0}{c-x_0}\right]^p \left[\frac{x_1-x}{x_1-c}\right]^q \ if\ x \in]x_0, x_1[\\ 0 \quad else \end{cases} \tag{1}$$

  Where $c = \frac{p\,x_1 + q\,x_0}{p+q}$ is the center of Beta function.

- **Case 2:** $p > 0,\ q = 0$

$$\beta(x) = \begin{cases} \left[\frac{x-x_0}{c-x_0}\right]^p \quad if \quad x \in]x_0, x_1[\\ 0 \quad if \quad x < x_0 \\ 1 \quad if \quad x > x_1 \end{cases} \tag{2}$$

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

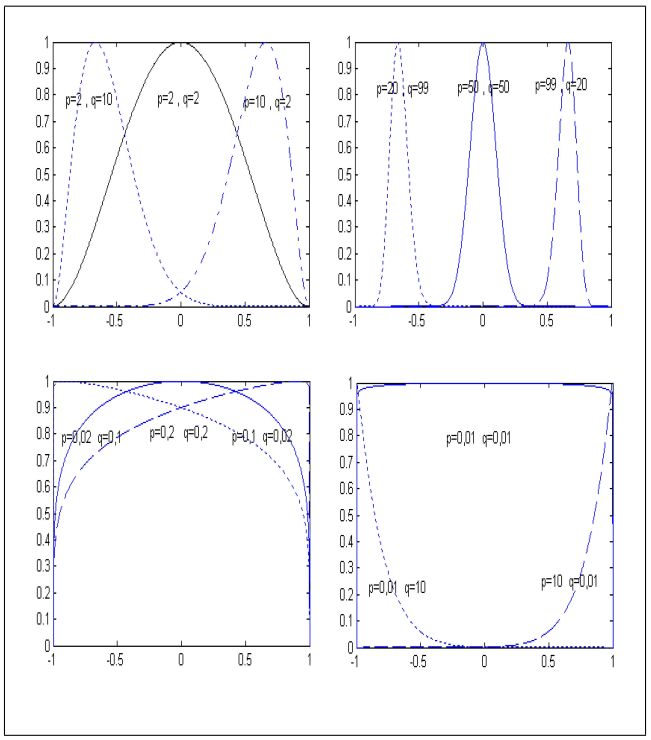

Fig. 1. Examples of Beta Basis Function.

- **Case 3:** $p = 0,\ q > 0$

$$\beta(x) = \begin{cases} \left[\frac{x_1 - x}{x_1 - c}\right]^q & if \quad x \in]x_0, x_1[\\ 1 \quad if \quad x < x_0 \\ 0 \quad if \quad x > x_1 \end{cases} \tag{3}$$

- **Case 4:** $p = 0,\ q = 0$

$$\beta(x) = 1, \quad \forall \quad x \in \mathbb{R} \tag{4}$$

Some proprieties taken from [18] in the one-dimensional case are presented as follows:

$$\beta(x_0) = \beta(x_1) = 0 \tag{5}$$

$$\beta(c) = 1 \tag{6}$$

$$\frac{d\beta(x)}{dx} = \left[\frac{px_1 + qx_0 - (p+q)x}{(x - x_0)(x_1 - x)}\right] * \beta(x) \tag{7}$$

$$\frac{d\beta(c)}{dx} = \frac{d\beta(x_0)}{dx} = \frac{d\beta(x_1)}{dx} = 0 \tag{8}$$

$$\frac{p}{q} = \frac{c - x_0}{x_1 - c} \tag{9}$$

We notice that if $p = 1,\ q = 0$:

$$\beta(x) = \begin{cases} \left[\frac{x - x_0}{c - x_0}\right] \quad if \quad x \in]x_0, x_1[\\ 0 \quad if \quad x < x_0 \\ 1 \quad if \quad x > x_1 \end{cases} \tag{10}$$

So, Beta basis function may be considered as a piecewise linear function of x, if $(p = 1, q = 0)$ or $(p = 0, q = 1)$.

Let $\sigma = x_1 - x_0$ is the width of the Beta function which can be seen as a scale factor for the distance $\| x - c \|$. So:

$$\begin{cases} x_0 = c - \left[\frac{\sigma p}{p+q}\right] \\ x_1 = c + \left[\frac{\sigma q}{p+q}\right] \end{cases} \tag{11}$$

(1) and (11) $\Rightarrow$

$$\begin{aligned} \beta(x) \quad &= \quad \beta(x; c, \sigma, p, q) \\ &= \begin{cases} \left[1 + \frac{(p+q)(x-c)}{\sigma p}\right]^p \left[1 - \frac{(p+q)(c-x)}{\sigma q}\right]^q \\ \qquad if \quad x \in \left]c - \frac{\sigma p}{p+q}, c + \frac{\sigma q}{p+q}\right[\\ 0 \quad else \end{cases} \end{aligned} \tag{12}$$

B. Beta Function against Gaussian Function

The Gaussian function is defined by:

$$Gauss(x; \mu, \sigma) = exp\Big[-\frac{(x - \mu)^2}{2\sigma^2}\Big] \tag{13}$$

$$Gauss(\mu) = 1 \tag{14}$$

$$\frac{dGauss(x)}{dx} = \left[\frac{x - \mu}{\sigma^2}\right] * Gauss(x) \tag{15}$$

$$\varepsilon \to 0 \quad if \quad n \to 0. \tag{16}$$

Alimi demonstrated in [18] that the Gaussian function can be approximated by the Beta function. In fact, for any given Gaussian function $Gauss(x; \mu, \sigma)$ and for any given precision ε, there exists a Beta function $\beta(x; x_0, x_1, p, q)$ that approximates the Gaussian function with an error of less than ε:

$$\Big|\beta(x; x_0, x_1, p, q) - Gauss(x; \mu, \sigma)\Big| \leqslant \varepsilon \quad \forall x \in \mathbb{R}. \tag{17}$$

He noted also that the reverse is not true, since the Beta function can have forms richer than the Gaussian function (asymmetry, linearity, etc.).

III. Flexible Beta Basis Function Neural Tree System: FBBFNT

In the current study, we have adopted a tree-based encoding method for representing the Beta basis function neural network instead of the matrix-based encoding method as it is more flexible and gives a more modifiable and adjustable structure. We introduce in this section, the proposed model for design the Beta basis function neural network through some definitions, basic concepts and the corresponding mathematical model. The proposed model is named Flexible Beta Basis Function Neural Tree (FBBFNT) [9], [10], [11], [12].

A. Definition of Flexible Beta Basis Function Neural Tree

The FBBFNT is formed of a node set S representing the union of function node set F and terminal node set T.

$$\begin{aligned} S = F \cup T = \{/_M\} \cup \{\beta_{ni}^l \,/\, n \in \{2, ..., M\}, i \in \{1, ..., N_c\}, \\ l \in \{2, ..., (Nl - 1)\}\} \cup \{x_1, ..., x_D\} \end{aligned} \tag{18}$$

Where:

- $/_M$ is the root node and represents a linear transfer function. M is the maximum degree of the tree;

- β_{ni}^{l} $(n \in \{2, ..., M\}, i \in \{1, ..., N_c\}, l \in \{2, ..., (Nl - 1)\})$ denote non-terminal hidden nodes and represent flexible Beta basis neurons with n inputs. i is the index of the node β with n inputs, N_c is the number of times in which β appears with n inputs. l is the layer index (this index is taken from top to bottom), and $Nl = depth$ is the number of layers (or the depth) of the tree;
- $x_1, x_2, ..., x_D$ are terminal nodes and define the input elements; each element includes LE learning values x_i^k $(i = 1, ..., D;\ k = 1, ..., LE)$; where D is the dimension of the treated problem and LE is the number of learning examples.

The output of a non-terminal node is calculated as a flexible neuron model (see Fig. 2).

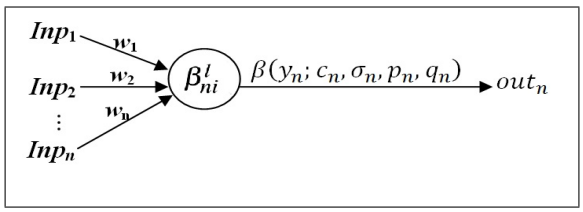

Fig. 2. A flexible neuron Beta operator.

In the creation process of flexible Beta basis function neural tree, if a function node, i.e., β_{ni}^{l} is selected, n real values are randomly created to represent the connection weights between the selected node and its offspring.
In addition, seen that the flexible transfer function used for the hidden layer nodes is the Beta function, four adjustable parameters (the center c_n, the width σ_n and the form parameters p_n, q_n) are randomly generated as flexible Beta operator parameters. For each non-terminal node, its total excitation is calculated by:

$$y_n = \sum_{j=1}^{n} w_j * Inp_j \tag{19}$$

Where Inp_j $(j = 1, ..., n)$ are the inputs of the selected node and w_j $(j = 1, ..., n)$ are the connection weights.
The Inp is formed by n points (Inp_j / $j \in \{1, ..., n\}$); each point includes LE learning values Inp_j^k $(k = 1, ..., LE)$. Each Inp_j can be either the values of a terminal node such that x_m $(m \in [1, D])$, or the output of another Beta node. The output of node β_{ni}^{l} (where $p_n > 0$, $q_n > 0$) is then calculated by:

$$out_n = \beta(y_n; c_n, \sigma_n, p_n, q_n) = \begin{cases} \left[1 + \frac{(p_n+q_n)(y_n-c_n)}{\sigma_n p_n}\right]^{p_n} \left[1 - \frac{(p_n+q_n)(c_n-y_n)}{\sigma_n q_n}\right]^{q_n} \\ \qquad if \quad y_n \in \left]c_n - \frac{\sigma_n p_n}{p_n+q_n}, c_n + \frac{\sigma_n q_n}{p_n+q_n}\right[\\ 0 \qquad else \end{cases} \tag{20}$$

The output layer yields a vector by linear combination of the node outputs of the last hidden layer to produce the final output.

A typical example of flexible Beta basis function neural tree model is shown in Fig. 3.

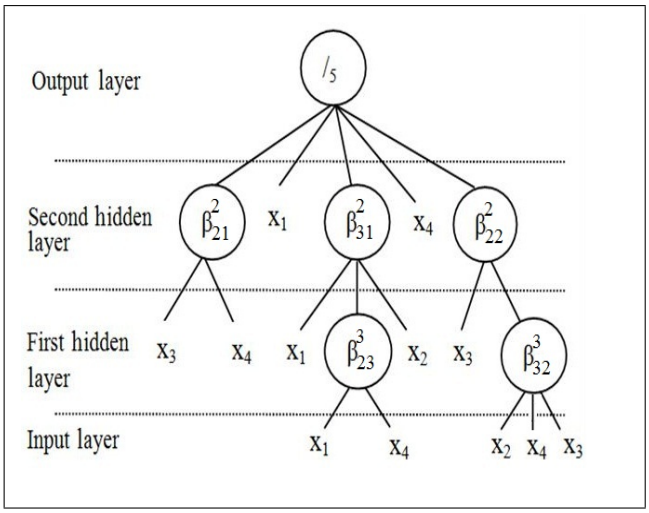

Fig. 3. A typical representation of FBBFNT: function node set $F = \{\beta_{21}^2, \beta_{22}^2, \beta_{31}^2, \beta_{23}^3, \beta_{32}^3, /_5\}$, and terminal node set $T = \{x_1, x_2, x_3, x_4\}$.

B. Basic concepts of FBBFNT

According to the definitions presented in the last part, the generation of the Flexible Beta Basis Function Neural Tree model is realized with a random and recursive way. More precisely, the initial FBBFNT model is a variable architecture neural network with uniformly distributed random number of layers in [*NLMin*, *NLMax*] and uniformly distributed random number of nodes for each layer in [*NNMin*, *NNMax*]; apart the output layer which has only one node. *NLMax* (the maximum layer number or the maximum depth of the tree) and *NNMax* (the maximum node number or the maximum degree of the tree) are chosen depending on the studied problem; unlike *NLMin* and *NNMin* which are equal to 3 and to 2, respectively. The minimum size of the tree is equal to 5 and its maximum size is calculated as following:

$$SizeMax = \frac{NNMax^{NLMax} - 1}{NNMax - 1} \tag{21}$$

The function nodes which contain parameters (Beta parameters and connection weights with the offspring nodes) are also randomly generated in the search spaces. Indeed, the connection weights vary between 0 and 1 (*Rand*[0, 1]), Beta center is in $[min(x), max(x)]$, Beta width is in $[0, |max(x) - min(x)|]$, and Beta form parameters (p, q) are in $[0, 5]$. The FBBFNT is encoded using three computer records.

BetaPar=**Record** {Computer record of Beta parameters}
c: integer; {center of the Beta}
σ: integer; {width of the Beta}
p: integer; {first form parameter of the Beta}
q: integer; {second form parameter of the Beta}
End;

Node=**Record** {Computer record of one node}
$parent$: integer; {value of parent node}
$type$: integer; {type of the node}
$value$: integer; {value of the node}
$NbChild$: integer; {number of children nodes}
$NLayer$: integer; {layer number}
$betaPar$: BetaPar; {record of Beta parameters}
$child$: array[1..NbChild] integer; {child indexes}
$weight$: array[1..NbChild] float; {weights}
End;

Tree=**Record** {The computer record of FBBFN tree}
$depth$: integer; {number of layers}
$size$: integer; {number of nodes}
$node$: array[SizeMin..SizeMax] Node;
End;

The overall output of flexible Beta basis function neural tree can be computed recursively again by depth-first method from left to right. Calculating the FBBFNT output depends on the type of nodes (function node or terminal node) and its level (root node or hidden node). The function used to calculate the FBBFNT output is given by algorithm 1.

Algorithm 1: Tree output computing algorithm.

```
[tree,out_tree]=Tree_Output(tree, CN, data)
/* Initial CN=1 */
input : tree: generated tree, CN: number of the current
        node, data: input data
output: tree: updating tree, out_tree: output of FBBFNT
begin
    if (tree.node[CN].type =1) then        /* CN is
    Function node */
        S = 0;
        for i := 1 to tree.node[CN].NbChild do
            [tree, out_child[i]] := Tree_Output (tree,
            tree.node[CN].child[i], data);
            S := S + tree.node[CN].weight[i] *
            out_child[i];
        end
        if (CN = 1) then          /* Root Node */
            out_tree := purelin(S);        /* Linear
            function */
        else
            out_tree :=
            BetaFunct(tree.node[CN].betaPar,S);
            /* Beta function */
        end
    else        /* CN is a terminal node */
        out_tree := data(tree.node[CN].value);
        /* corresponded input vector */
    end
end
```

C. Computational complexity of FBBFNT

The computational complexity of the FBBFNT model can be described as follows. Let us assume that N is the number of input data, so the computational complexity of the FBBFNT model is as a function of this number. For the FBBFNT's generation algorithm and output computing algorithm which are recursive algorithms, the complexity is of the order of $O(N\ log_M(N))$; with M is the maximum degree of the tree. On the other hand, the structure optimization algorithms are in the worst case, of the order of $O(N^2\ log_M(N))$ and the parameter optimization algorithms are in the worst case, of the order of $O(N^2)$. Moreover, the complexity of fitness function is of the order of $O(N)$.

As conclusion, the computational complexity of the FBBFNT program is almost cubic, i.e. $O(N^3\ log_M(N))$.

D. Mathematical model of FBBFNT

We can then consider that Tree_Output is a mathematical function noted by: $g(x)$ where x is input data and it defined as follow:

- **Case 1**: $j = depth$

$$g^{depth}(x) = \sum_{i=1}^{Nbchild_{depth}} w_i g_i^{depth-1}(x) \tag{22}$$

- **Case 2**: $1 \leq j < depth$

$$g_i^j(x) = \begin{cases} \beta(\sum_{k=1}^{Nbchild_i} w_k g_k^{j-1}(x)) \\ \qquad if(type_i = 1)\ or\ (1 < j < depth) \\ x_m\ \forall m \in [1, D] \\ \qquad if(type_i = 2)\ or\ (j = 1) \end{cases} \tag{23}$$

Where j is the layer-associated node number and i is the child nodes' index.

IV. Universal Approximation of FBBFNT

To prove theoretically the competence of the proposed model in problem approximation, we will demonstrate, in this section, that the Flexible Beta Basis Function Neural Tree is a universal approximator. In fact, Hornik *et al.* have proved in [19], that the multilayer feed-forward networks are universal approximators. Since FBBFNT is a special case of multilayer feed-forward networks, we can deduce that FBBFNT possesses this property especially that Alimi *et al.* [3] have also justified that BBFNN is a universal approximator. We will justify, moreover, mathematically this property in this section.

Definition 1: Let A be a set of elements and Ψ be a set of scalars.
A has two internal composition laws "+" (addition between elements of A), "$\times$" (multiplication between elements of A), and an external law of composition "." (Multiplication of an element of A by a scalar of Ψ), A is algebra if and only if:

1) A endowed with the composition law "+" and the external law of composition ".", is a linear space.

2) If f, g and h are in A, α is in Ψ, then:
   - $f \times g$ is an element of A.
   - $(f \times g) \times h$ is an element of A.
   - $f \times (g + h) = (f \times g) + (f \times h)$ is an element of A.
   - $\alpha.(f \times g) = (\alpha.f) \times g$.

Definition 2: a set B is sub-algebra of the algebra A if:
1) B is a linear subspace of A.
2) If f and g are in B, then $f \times g$ is an element of B.

Theorem 1 (Stone-Weierstrass theorem) [20]:
Let X be a metric space, X is a compact.
Let $C[X]$ be a set of continuous functions defined on X and let A sub-algebra of $C[X]$ satisfying the following properties:
1) The function $f(x) = 1$ belongs to A.
2) For any pair (x, y) in $X \times X$ such that $x \neq y$, there exists a function $f \in A$ such that $f(x) \neq f(y)$

Then A is dense in $C[X]$.

Theorem 2 (The FBBFNT density): Let X be a compact in $\mathbb{R}^D$, and:

$$T_1 = \{f \in C[X] \ \ / f(x) = \sum_{i=1}^{m} w_i g_i(x), m \in \mathbb{N}\} \quad (24)$$

Where $g_i(x)$ represent the function defined in the equation 22.
Then T_1 is dense in $C[X]$.

***Proof*:**

1) For example: if $depth = 3$ and $NbChild_{depth} = 2$, then the root node has two child nodes which are Beta function nodes. Also if $p_1 = p_2 = 0$ and $q_1 = q_2 = 0$, then $\beta_1(x) = \beta_2(x) = 1, \forall x \in \mathbb{R}$; So according to equations 4, 22 and 23:

$$f(x) = g^{depth}(x) = \sum_{i=1}^{2} w_i \beta_i(x) = \sum_{i=1}^{2} w_i = 1 \quad such\ that\ w_1 = w_2 = 0.5 \quad (25)$$

then $f(x) = 1$ belongs to T_1.

2) For any two distinct points s and r we can find a function f in T_1 such that $f(s) \neq f(r)$, Indeed: if $depth = 3$, $NbChild_{depth} = 2$, the root node has two children nodes which are Beta function nodes with:

$$\beta_1(x) = \beta_2(x) = \begin{cases} \left(\frac{x-x_0}{c-x_0}\right)^p \left(\frac{x_1-x}{x_1-c}\right)^q & if x \in]x_0, x_1[\\ 0 \quad else \end{cases} \quad (26)$$

$p_1 = p_2$ and $q_1 = q_2$ are chosen such that $c = s$, then $\beta(s) = 1$ and $\beta(r) < 1$
So $f(x) = \sum_{i=1}^{2} w_i \beta_i(x)$, then $f \in T_1$, $f(s) = 1$ and $f(r) \neq 1 \Rightarrow f(s) \neq f(r)$.

3) We must show that T_1 is sub-algebra of $C[X]$ for any compact X in $\mathbb{R}^D$.
T_1 is sub-algebra of $C[X]$ if the product of two of these elements gives us another element of T_1 and if it is a subspace of $C[X]$.

- Since T_1 is a linear superposition of the function g as defined in the equation 23, it is sufficient to show that the product of two g functions is an element of T_1. Indeed:

$$g_i^j(x) = \beta\Big(\sum_{k=1}^{Nbchild_i} w_k g_k^{j-1}(x)\Big) = \beta \circ h_i^j(x) = \beta(y_i) \quad (27)$$

with:
$h_i^j(x) = y_i = \sum_{k=1}^{Nbchild_i} w_k g_k^{j-1}(x)$;

- Case 1: g_1 is non-terminal node and g_2 is terminal node

$$g_1^{depth-1}(x) = \beta \circ h_1^{depth-1}(x) = \beta(y_1) = \left(\frac{y_1 - x_0}{c_1 - x_0}\right)^{p_1} \left(\frac{x_1 - y_1}{x_1 - c_1}\right)^{q_1} \quad (28)$$

$$g_2^{depth-1}(x) = x_m \ \ / \ \ m \in [1, D] \quad (29)$$

So:
$h(x) = g_1^{depth-1}(x) * g_2^{depth-1}(x)$
$= x_m \beta(y) \Rightarrow h(x) \in T_1$

- Case 2: g_1 is non-terminal node and g_2 is also non-terminal node

$$g_1^{depth-1}(x) = \beta \circ h_1^{depth-1}(x) = \beta(y_1) = \left(\frac{y_1 - x_0}{c_1 - x_0}\right)^{p_1} \left(\frac{x_1 - y_1}{x_1 - c_1}\right)^{q_1} \quad (30)$$

$$g_2^{depth-1}(x) = \beta(y_2) = \left(\frac{y_2 - x_0}{c_2 - x_0}\right)^{p_2} \left(\frac{x_1 - y_2}{x_1 - c_2}\right)^{q_2} \quad (31)$$

So, we have,

$$h(x) = g_1^{depth-1}(x) * g_2^{depth-1}(x) = \beta(y_1)\beta(y_2) = \left(\frac{y_1-x_0}{c_1-x_0}\right)^{p_1} \left(\frac{x_1-y_1}{x_1-c_1}\right)^{q_1} \left(\frac{y_2-x_0}{c_2-x_0}\right)^{p_2} \left(\frac{x_1-y_2}{x_1-c_2}\right)^{q_2}$$

With:
$y_1 = \left(\frac{x-x_0}{c_3-x_0}\right)^{p_3} \left(\frac{x_1-x}{x-c_3}\right)^{q_3}$, and

$y_2 = \left(\frac{x-x_0}{c_4-x_0}\right)^{p_4} \left(\frac{x_1-x}{x-c_4}\right)^{q_4}$

So for example:
$\left(\frac{y_1-x_0}{c_1-x_0}\right)^{p_1} = E$
$= \left(\left(\left(\frac{x-x_0}{c_3-x_0}\right)^{p_3}\left(\frac{x_1-x}{x_1-c_3}\right)^{q_3}\right)-x_0\right)^{p_1} * \left(\frac{1}{c_1-x_0}\right)^{p_1}$

By polynomial development:
$E = \sum_{k=0}^{p_1}\left[\left(\prod_{i=1}^{k}\frac{p_1-(k-i)}{i}\right)\ x_0^k \left(\frac{x-x_0}{c_3-x_0}\right)^{p_3(p_1-k)}\left(\frac{x_1-x}{x-c_3}\right)^{q_3(p_1-k)}\right]$

$E = \sum_{k=0}^{p_1}\left[\left(\prod_{i=1}^{k}\frac{p_1-(k-i)}{i}\right)\ x_0^k \beta_k(x; x_0, x_1, p_3(p_1-k), q_3(p_1-k))\right]$

And by the same principle we can calculate all terms of $h(x)$ and assume that $h(x)$ is a Beta function with specific parameters.

- Let: $f(x) = \sum_{i=1}^{m_1} w_i g_i^1(x)$ and

  $k(x) = \sum_{i=1}^{m_2} w_i g_i^2(x)$ two elements of T_1; where $m_1 \in \mathbb{N}$, $m_2 \in \mathbb{N}$ and λ is a scalar.

  Let: $\Omega_1 = \{g_1^1, ..., g_{m_1}^1\}$, $\Omega_2 = \{g_1^2, ..., g_{m_2}^2\}$ and $\Omega = \{g_1, ..., g_m\}$ with: $\Omega = \Omega_1 \bigcup \Omega_2$

  Then, we can write f and k as follow:

  $f(x) = \sum_{i=1}^{m} \alpha_i g_i(x)$ and $k(x) = \sum_{i=1}^{m} \gamma_i g_i(x)$

  So:
  $f(x) + \lambda k(x) = \sum_{i=1}^{m} \alpha_i g_i(x) + \lambda \sum_{i=1}^{m} \gamma_i g_i(x)$

  $= \sum_{i=1}^{m} (\alpha_i + \lambda\gamma_i) g_i(x)$

  Thus: $f + \lambda k \in T_1$

We therefore conclude that T_1 is dense in $C[X]$.

Theorem 3 (Approximation propriety of FBBFNT):
Let X be a compact in $\mathbb{R}^D$, and:

$$T_1 = \{f \in C[X] \ \ / f(x) = \sum_{i=1}^{m} w_i g_i(x), m \in \mathbb{R}^D\} \quad (32)$$

Where $g_i(x)$ represent the function defined in the equation 22.
Then: $\forall f \in C[X], \forall \varepsilon > 0, \exists g \in T_1$ such that $\| f-g \|_\infty \leq \varepsilon$.

Proof: (Evident by applying Theorem 2)

V. Simulation Results

FBBFNT needs a evolution process in order to optimize its structure and to adjust its parameters (Beta parameters and connection weights). So, to find an optimal or near-optimal FBBFNT model, structure and parameter optimization are used simultaneously in a hybrid algorithm. As our previous work in [12], the FBBFNT's structure is generated and evolved by the Extended Immune Programming (EIP) algorithm and the FBBFNT's parameters are optimized using Hybrid Bacterial Foraging Optimization Algorithm (HBFOA).

The evolving FBBFNT model is applied to approximate the input/output map of nonlinear systems. Indeed, in order to prove the effectiveness of FBBFNT model, we compare its results with those provided by other learning methods in the literature.

A. Example 1: Approximation of Mackey–Glass time series

A time-series approximation problem can be constructed based on the Mackey-Glass [21] differential equation:

$$\frac{dx(t)}{dt} = \frac{a\ x(t-\tau)}{1+x^c\ (t-\tau)} - b\ x(t) \quad (33)$$

The settings of the experiment vary from one work to another. In our case, we take $a = 0.2$, $b = 0.1$, $c = 10$, and $\tau = 17$. These values are the same ones used by the comparison systems [22], [23], [24], [25], [26], [27], [28], [29]. As in the studies mentioned above, the task is to predict the value of the time series at point $x(t+6)$, with using the inputs variables $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$. 1000 sample points are used in our study. The first 500 data pairs of the series are used as training data, while the remaining 500 are used to validate the model identified.

The used node set for creating an optimal FBBFNT model is $S = F \cup T = \{\beta_{21}^3, \beta_{22}^3, \beta_{31}^2, /_3\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i (i = 1, 2, 3, 4) denotes $x(t)$, $x(t-6)$, $x(t-12)$ and $x(t-18)$, respectively. After 16 generations ($G = 16$) and 6,004,148 global number of function evaluations of the hybrid learning algorithm, an optimal FBBFNT model was obtained with RMSE $5.3430e-10$. The RMSE value for validation data set is $1.8630e-09$. These important results are explained in the adaptation of the tree representation which is a universal approximator, on the one hand and in the application of EIP and HBFOA algorithms for the model evolution, on the other hand.

The proposed system is essentially compared with Hierarchical multi-dimensional differential evolution for the design of Beta basis function neural network (HMDDE-BBFNN) [22], the FNT model with Gaussian function as flexible neuron operator [26], the local least-squares support vector machines-based neuro-fuzzy model (LNF) [29] and also with other systems.
The HMDDE-BBFNN approach adopts for parameters: 50 for the population size, 10,000 for a total number of iterations, and 4 for the number of the hidden nodes. Moreover, the parameter settings of the FNT system [26] are 30 to the population size, 135 as generation number, and 4 as hidden function unit number (with two hidden layers). For LNF network, the authors use 6 neurons to generate their model.

The comparison results are shown in Table I. As observed, the FBBFNT_EIP&HBFOA achieves the lowest training and testing errors.

TABLE I

COMPARISON FBBFNT_EIP&HBFOA WITH OTHER METHODS FOR THE MACKEY-GLASS TIME-SERIES.

Method	RMSE Training	RMSE Testing
HMDDE-BBFNN [22]	0.0094	0.0170
GA-BBFNN [23]	-	0.013
Fuzzy&MRB [24]	0.000990	0.000884
CPSO [25]	0.0199	0.0322
FNT [26]	0.0069	0.0071
HCMSPSO [27]	0.0095	0.0208
FWNN-M [28]	0.00129	0.00114
LNF [29]	0.00070	0.00079
FBBFNT_EIP&HBFOA	**5.3430e-10**	**1.8630e-09**

B. Example 2: Approximation of Sunspot Number time series

This example presents the series of the sunspot annual average numbers which show the yearly average relative number of sunspot observed. The sunspot number time series is considered as a real-world highly-complex and non-stationary time series [30]. It is recorded for the years 1700-1979. The dataset is available at the National Geophysical Data Center website (http://www.ngdc.noaa.gov/stp/solar/ssndata.html).

The data points between 1700 and 1920 are used for training FBBFNT model. For the test two sets are used the first one is from 1921 to 1955 and the second is from 1956 to 1979. The $y(t-4)$, $y(t-3)$, $y(t-2)$ and $y(t-1)$ are used as inputs to the FBBFNT model in order to predict the output y(t). The used node set for the FBBFNT model is $S = F \cup T = \{\beta_{21}^3, \beta_{22}^3, \beta_{31}^2, /_3\} \cup \{x_1, x_2, x_3, x_4\}$, where x_i (i = 1, 2, 3, 4) denotes $y(t-4)$, $y(t-3)$, $y(t-2)$ and $y(t-1)$, respectively. After 26 generations of the evolution (G = 26), an optimal FBBFNT model was obtained with RMSE $1.9566e-10$. The RMSE value for the first data set validation is $4.1519e-10$ and for the second data set validation is $7.2714e-10$.

Table II illustrates the comparison of the proposed algorithm with other models according to the training and testing errors. As evident from Table II, FBBFNT_EIP&HBFOA shows again the efficiencies for the sunspot number time series.

TABLE II

COMPARISON FBBFNT_EIP&HBFOA WITH OTHER METHODS FOR THE SUNSPOT NUMBER TIME-SERIES.

Method	RMSE Training	RMSE Testing 1	RMSE Testing 2
Transversal Net [31]	0.2653	0.3149	1.1349
Recurrent Net [31]	0.2679	0.3151	0.9005
RFNN [32]	-	0.2749	0.6249
FWNN-S [33]	0.2527	0.3341	0.5299
FWNN-R [33]	0.2383	0.3350	0.6885
FWNN-M [33]	0.2430	0.3152	0.6080
ABC_BBFNN [34]	0.0012	0.0018	0.0044
LNF [29]	0.1888	0.2537	0.3808
FBBFNT_EIP&HFBOA	**1.9566e-10**	**4.1519e-10**	**7.2714e-10**

C. Example 3: Approximation of Box and Jenkins' gas furnace time series

The gas furnace data of Box and Jenkins [35] was saved from a combustion process of a methane-air mixture. It is used as a benchmark example for testing approximation methods. The data set forms of 296 pairs of input-output measurements. The input $u(t)$ is the gas flow into the furnace and the output $y(t)$ is the CO_2 concentration in outlet gas. The inputs for constructing FBBFNT model are $y(t-1)$, $u(t-4)$, and the output is $y(t)$. In this study, 200 data samples are used for training and the remaining data samples are used for testing the performance of the proposed model. The used instruction set is $S = F \cup T = \{\beta_{21}^2, \beta_{22}^3, /_3\} \cup \{x_1, x_2\}$, where x_i (i = 1, 2) denotes $y(t-1)$, $u(t-4)$, respectively. After 22 generations ($G = 22$) of the learning algorithm, the optimal FBBFNT model was obtained with the RMSE 0.008026. The RMSE value for validation data set is 0.009121. A comparison result of different methods for Jenkins-Box data approximation is shown in Table III.

TABLE III

COMPARISON FBBFNT_EIP&HBFOA WITH OTHER METHODS FOR THE JENKINS-BOX TIME-SERIES ($y(t-1)$, $u(t-4)$).

Method	RMSE Training	RMSE Testing
ANFIS model [36]	-	0.08544
FuNN model [37]	-	0.26720
FNT [26]	0.01705	0.01746
FWNN-M [28]	0.01963	0.02324
HMDDE-BBFNN [22]	0.3745	0.2411
HyFIS model [38]	-	0.25245
FBBFNT_EIP&HBFOA	**0.008026**	**0.009121**

VI. CONCLUSIONS

In this paper, the universal approximation is demonstrated for the Flexible Beta Basis Neural Tree (FBBFNT). The FBBFNT is a particular case of multilayer artificial neural network using tree-based encoding method. This propriety needs firstly to prove the FBBFNT density' theorem based on the density theorem of Stone [20].

The experiment results show that the FBBFNT network can effectively approximate the time-series problems such as the Mackey-Glass chaotic time series, the Sunspot Number time series, and the Jenkins-Box time series.

ACKNOWLEDGMENT

The authors would like to acknowledge the financial support of this work by grants from General Direction of Scientific Research (DGRST), Tunisia, under the ARUB program. This work was also supported in the framework of the IT4 Innovations Centre of Excellence project, reg. no. CZ.1.05/1.1.00/02.0070 by operational programme 'Research and Development for Innovations' funded by the Structural Funds of the European Union and state budget of the Czech Republic, EU.

REFERENCES

[1] Adel M. Alimi, "The Beta Fuzzy System: Approximation of Standard Membership Functions", *17ème Journées Tunisiennes d'Electrotechnique et d'Automatique, JTEA'97*, pp. 108-112, Nabeul, Tunisia, 1997.

[2] Adel M. Alimi, "The Beta System: Toward a Change in Our Use of Neuro-Fuzzy Systems", *International Journal of Management*, Invited Paper, pp. 15-19, June 2000.

[3] Adel M. Alimi, R. Hassine and M. Selmi, "Beta fuzzy logic systems: Approximation properties in the MIMO case", *International Journal of Applied Mathematics and Computer Science*, vol. 13, no. 2, pp. 225-238, 2003.

[4] C. Aouiti, Adel M. Alimi and A. Maale, "Genetic Algorithms to Construct Beta Neuro- Fuzzy Systems", *International Conference on Computational & Artificial Intelligence for Decision, Control and Automation, ACIDCA'2000*, pp. 88-93, 2000.

[5] Adel M. Alimi, "The recognition of arabic handwritten characters with the Beta neuro-fuzzy network, *17ème Journées Tunisiennes d'Electrotechnique et d'Automatique*, JTEA'97, vol. 1, pp. 349-356, Nabeul, Tunisia, 1997.

[6] Adel M. Alimi, "On-Line Analysis of Handwritten Arabic: A New Approach to Recognize Segmented Characters from Cursive Script", *Les Annales Maghrébines de l'Ingénieur*, vol. 14, no. 1, pp. 7-27, 2000.

[7] H. Dhahri, Adel M. Alimi and F. Karray, "Designing beta basis function neural network for optimization using particle swarm optimization", *International Joint Conference on Neural Networks*, pp. 2564-2571, Hong Kong, China, 1-6 June 2008.

[8] H. Dhahri and Adel M. Alimi, "Opposition-based differential evolution for beta basis function neural network", *IEEE Congress on Evolutionary Computation*, pp. 1-8, China, 18-23 July 2010.

[9] S. Bouaziz, H. Dhahri and Adel M. Alimi, "Evolving flexible beta operator neural trees (FBONT) for time series forecasting", *the 19th international conference on Neural Information Processing - Volume Part III, ICONIP'12, Springer-Verlag*, pp. 17-24, Doha, Qatar, 2012.

[10] S. Bouaziz, H. Dhahri, Adel M. Alimi and A. Abraham, "A Hybrid Learning Algorithm For Evolving Flexible Beta Basis Function Neural Tree Model", Neurocomputing, vol. 117, pp. 107-117, 2013.

[11] S. Bouaziz, Adel M. Alimi and A. Abraham, "Extended Immune Programming and Opposite-based PSO for Evolving Flexible Beta Basis Function Neural Tree", *IEEE International Conference on Cybernetics*, Lausanne Switzerland, pp. 13-18, 13-15 June 2013.

[12] S. Bouaziz, Adel M. Alimi and A. Abraham, "Evolving Flexible Beta Basis Function Neural Tree for nonlinear systems", *International Joint Conference on Neural Networks*, Dallas Texas, 4-9 August 2013.

[13] M. Ammar, S. Bouaziz, Adel M. Alimi and A. Abraham, "Hybrid Harmony Search algorithm for Global Optimization", *Fifth World Congress on Nature and Biologically Inspired Computing*, pp. 69-75, Fargo-USA, 12-14 August 2013.

[14] Y. Jarraya, S. Bouaziz, Adel M. Alimi and A. Abraham, "The Adaptive Chemotactic Foraging with Differential Evolution algorithm", *Fifth World Congress on Nature and Biologically Inspired Computing*, Fargo-USA, pp. 63-68, 2013.

[15] Y. Jarraya, S. Bouaziz, Adel M. Alimi and A. Abraham, "A Hybrid Computational Chemotaxis in Bacterial Foraging Optimization Algorithm for Global Numerical Optimization", *IEEE International Conference on Cybernetics*, Lausanne Switzerland, pp. 213-218, 2013.

[16] J. Murata, M. Suzuki and K. Hirasawa, "Networks with Input Gates for Situation-Dependent Input Selection in Reinforcement Learning", Proceedings of the International Joint Conference on Neural Networks, IJCNN'02, pp. 5-10, 2002.

[17] Adel M. Alimi, "What are the advantages of using the Beta neuro-fuzzy system?", *IEEE/IMACS Multiconf. Computational Engineering in Systems Applications, CESA'98*, vol. 2, pp. 339-344, Hammamet, Tunisia, 1998.

[18] Adel M. Alimi, "Beta Neuro-Fuzzy Systems", *TASK Quarterly Journal, Special Issue on "Neural Networks"*, vol. 7, no. 1, pp. 23-41, 2003.

[19] K. Hornik, M. Stinchcombe and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, no. 5, pp. 359-366, July 1989.

[20] M. H. Stone, "Applications of the Theory of Boolean Rings to General Topology", *Transactions of The American Mathematical Society*, vol. 41, 1937.

[21] L. Glass and M. C. Mackey, "Pathological physiological conditions resulting from instabilities in physiological control systems", *Annals of the New York Academy of Sciences*, vol. 316, pp. 214-235, 1979.

[22] H. Dhahri, Adel M. Alimi and A. Abraham, "Hierarchical multidimensional differential evolution for the design of beta basis function neural network", *Neurocomputing*, vol. 97, pp. 131-140, 2012.

[23] C. Aouiti, Adel M. Alimi and A. Maalej, "A Genetic Designed Beta Basis Function Neural Networks for approximating of multi-variables functions", *Int. Conf. Artificial Neural Nets and Genetic Algorithms, Springer Computer Science*, pp. 383-386, Prague, Czech Republic, 2001.

[24] C.G. Coy and D. Kaur, "Improving evolutionary training for Sugeno Fuzzy Inference Systems using a Mutable Rule Base", *Annual Meeting of the North American, Fuzzy Information Processing Society (NAFIPS)*, pp. 1-6, 12-14 July, 2010.

[25] F. V. D Bergh and A. P. Engelbrecht, "A Cooperative approach to particle swarm optimization", *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225-239, 2004.

[26] Y. Chen, B. Yang, J. Dong and A. Abraham, "Time-series forecasting using flexible neural tree model", *Information Sciences*, vol. 174, no. 3-4, pp. 219-235, August 2005.

[27] C-F. Juang, C-M. Hsiao and C-H. Hsu, "Hierarchical cluster-based multispecies particle-swarm optimization for fuzzy-system optimization", *IEEE Transactions on Fuzzy Systems*, vol. 18, no. 1, pp. 14-26, February 2010.

[28] S. Yilmaz and Y. Oysal, "Fuzzy wavelet neural network models for prediction and identification of dynamical systems", *IEEE Transactions on Neural Networks*, vol. 21, no. 10, pp. 1599-1609, October 2010.

[29] A. Miranian and M. Abdollahzade, "Developing a Local Least-Squares Support Vector Machines-Based Neuro-Fuzzy Model for Nonlinear and Chaotic Time Series Prediction", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 2, pp. 207-218, 2013.

[30] A. J. Izeman , J.R. Wolf and the Zurich, "sunspot relative numbers", *The Mathematical Intelligence*, vol.7, no.1, pp. 27-33, 1985.

[31] J.R. McDonnell, D. Waagen, "Evolving recurrent perceptrons for time-series modeling", *IEEE Trans Neural Networks*, vol. 5(1), pp. 24-38, 1994.

[32] R.A. Aliev, B.G. Guirimov, R.R. Aliev, "Evolutionary algorithm-based learning of fuzzy neural networks", *Part 2: Recurrent fuzzy neural networks, Fuzzy Sets and Systems*, vol. 160 (17), 2009.

[33] S. Yilmaz, Y. Oysal, "Fuzzy wavelet neural network models for prediction and identification of dynamical systems", *IEEE Transactions on Neural Networks*, pp. 1599-1609, 2010.

[34] H. Dhahri, Adel M. Alimi, "Designing Beta Basis Function Neural Network for Optimization Using Artificial Bee Colony (ABC)", International Joint Conference on Neural Networks, Brisbane, pp. 2161-4393, 2012.

[35] G.E.P. Box and G.M. Jenkins, "Time series analysis: forecasting and control", *Holden-Day series in time series analysis and digital processing*, 1976.

[36] J. Nie, "Constructing fuzzy model by self-organising counter propagation network", *IEEE Transactions on Systems Man and Cybernetics*,vol. 25, pp. 963-970, 1995.

[37] J.-S.R. Jang, C.-T. Sun, E. Mizutani, "Neuro-fuzzy and soft computing: a computational approach to learning and machine intelligence", *Prentice-Hall*, Upper Saddle River, NJ, 1997.

[38] K. K. Nikola, K. Jaesoo, J. W. Michael and R. G. Andrew, "FuNN/2 - A Fuzzy Neural Network Architecture for Adaptive Learning and Knowledge Acquisition", *Information Sciences*, vol. 101, no. 3-4, pp. 155-175, 1997.