University of Ostrava

Institute for Research and Applications of Fuzzy Modeling

Fuzzy transform from a neural network point of view

Martin Štěpnička and Ondřej Polakovič

Research report No. 96

2006

Submitted/to appear:

IPMU '06

Supported by:

Project MSM6198898701 of the MŠMT ČR; Project DAR No. 1M0572 of the MŠMT ČR

University of Ostrava
Institute for Research and Applications of Fuzzy Modeling
30. dubna 22, 701 03 Ostrava 1, Czech Republic

tel.: +420-59-6160234 fax: +420-59-6120 478
e-mail: martin.stepnicka@osu.cz ; ondrej.polakovic@osu.cz

ABSTRACT:

The paper deals with the F-transform technique which was introduced as a method for an approximate representation of continuous functions which is appropriate for many applications. In the terminology of neural networks, the F-transform uses so called off-line (batch) learning. However, for certain applications an on-line (incremental) learning algorithm has to be implemented. We study the F-transform from a neural network point of view and introduce an on-line learning based on the gradient descent method. Moreover, we introduce an on-line algorithm tuning the fuzzy partition used in the F-transform method.

Keywords: Fuzzy set, F-transform, Neural network, RBF, Gradient descent method.

1 Motivation

Fuzzy approximation is a newly developing mathematical branch aiming at approximation of say some dependencies by means of the fuzzy set theory and the fuzzy logic in broader sense. Obviously it has old roots in Takagi-Sugeno fuzzy rule based systems and in works aiming at approximation capabilities of fuzzy rule based systems, see [2, 6].

The approximation task is in mathematics very and already well studied so, we hardly avoid building bridges between fuzzy approximation and already done results. Conversely, we can simply inherit many results from other branches dealing with the approximation problem.

This paper is an introduction of the study of different relationships between fuzzy approximation methods and other approximation techniques. It deals with a particular fuzzy approximation method called *fuzzy transform* (F-transform) and neural networks as another soft computing branch which has many times been proven to be an appropriate tool for approximation tasks.

By getting both approaches closer to each other we expect:

- development of new algorithms (known in neural networks) for fuzzy approximation
- enriching both branches by already done results from each other
- possible improvements
- answering natural question about similarities and similar problems in both branches
- inheriting theoretical results e.g. conditions of universal approximations etc.

At this first stage of our investigation, we simply try to look at the fuzzy transform problem from a neural network point of view to open this problematic, inherit neural algorithms, investigate possible improvements, implement an on-line type of learning and build a bridge between both branches for next theoretical results and algorithmic improvements.

2 Preliminaries

The F-transform technique has been introduced in [9]. Several theoretical result and successful practical applications have been published since the introduction, see [10, 11, 8, 16].

Basically, it is an approximation method which deals with a fuzzy partition of a domain of an approximated function. The direct F-transform transforms an approximated function to a discrete vector where every single component of the vector represents all values of the approximated function above a support of a respective fuzzy set from the fuzzy partition of the domain. The representation is supposed to be the best one, in some sense.

For a bit more detailed explanation let us recall basic facts and definitions about the F-transform.

Definition 1 Let $c_i = a + h(i-1)$ be nodes on $[a,b]$ where $h = (b-a)/(n-1)$, $n \geq 2$ and $i = 1, \ldots, n$. We say that functions $\mathbf{A}_1(x), \ldots, \mathbf{A}_n(x)$ defined on $[a,b]$ are *basic functions* if each of them fulfils the following:

- $\mathbf{A}_i : [a,b] \to [0,1]$, $\mathbf{A}_i(x_i) = 1$,

- $\mathbf{A}_i(x) = 0$ if $x \notin (c_{i-1}, c_{i+1})$, where $c_0 = a,\ c_{n+1} = b$,
- $\mathbf{A}_i(x)$ is continuous,
- $\mathbf{A}_i(x)$ strictly increases on $[c_{i-1}, c_i]$ and strictly decreases on $[c_i, c_{i+1}]$,
- $\sum_{i=1}^{n} \mathbf{A}_i(x) = 1$, for all $x \in [a, b]$,
- $\mathbf{A}_i(c_i - x) = \mathbf{A}_i(c_i + x)$, for all $x \in [0, h],\ i = 2, \ldots, n-1,\ n > 2$,
- $\mathbf{A}_{i+1}(x) = \mathbf{A}_i(x - h)$, for all $x \in [a + h, b],\ i = 2, \ldots, n-2,\ n > 2$.

It is easy to see, that the basic functions form a fuzzy partition (see [12]) of the given domain and each basic function $\mathbf{A}_i$ can be viewed as a fuzzy number 'approximately x_i'.

The basic functions can be generalized to the nonsymmetric ones, see [8]. In that case, we define $h_i = c_{i-1} - c_i$ and the basic functions $\mathbf{A}_i$ fulfill only the first five conditions from Definition 1.

Now, we consider a continuous function $f : [a, b] \to \mathbb{R}$ which is to be approximated by an element of a class of approximating functions given by linear combinations of the basic functions

$$f_n^F(x) = \sum_{i=1}^{n} F_i \mathbf{A}_i(x) \tag{1}$$

where $F_i \in \mathbb{R}$.

The real coefficients F_i are called the *components* of the F-transform and they are determined to minimize the following error function

$$E(Q_1, \ldots, Q_n) = \int_a^b \sum_{i=1}^{n} (f(x) - Q_i)^2 \mathbf{A}_i(x). \tag{2}$$

By a direct computation one can check that the components given by the following formula

$$F_i = \frac{\int_a^b f(x)\mathbf{A}_i(x)}{\int_a^b \mathbf{A}_i(x)}, \quad i = 1, \ldots, n \tag{3}$$

minimize error function (2), see [8, 10].

In most practical cases, we are not given an analytical description of the approximated function f but only some, say measured, samples $(x_k, f(x_k))$ where $k = 1, \ldots, K$ and in principle $n << K$. For these cases the error function is given as follows

$$E(Q_1, \ldots, Q_n) = \sum_{k=1}^{K} \sum_{i=1}^{n} (f(x) - Q_i)^2 \mathbf{A}_i(x) \tag{4}$$

and the components are given analogously

$$F_i = \frac{\sum_{k=1}^{K} f(x_k)\mathbf{A}_i(x_k)}{\sum_{k=1}^{K} \mathbf{A}_i(x_k)}, \quad i = 1, \ldots, n \tag{5}$$

as well.

The real vector $[F_1, \ldots, F_n]$ given by formula (3) or (5) is called the *direct F-transform*. Continuous approximating function $f_n^F(x)$ given by (1) is called the *inverse F-transform*.

The direct F-transform formula keeps the linearity condition i.e. let $[F_1, \ldots, F_n]$ be the direct F-transform of a function f, $[G_1, \ldots, G_n]$ be the direct F-transform of a function g and finally $[H_1, \ldots, H_n]$ be the direct F-transform of $h = \alpha f + \beta g$, $\alpha, \beta \in \mathbb{R}$ then

$$H_i = \alpha F_i + \beta G_i, \quad i = 1, \ldots, n. \tag{6}$$

Therefore we can talk about a transform. We simply transform a function into a real vector and then the vector is transformed back to the space of continuous functions. The necessary condition of the

uniform convergence of the sequence of the inverse F-transform for $h_i \to 0$ has been already proved, see e.g. [8].

The F-transform technique has been many times shown to be appropriate for a number of applications. Let us stress its main advantages and properties: linearity and uniform convergence [8, 10], computational simplicity and fastness [8, 15], smoothing abilities [11, 15, 16], noise removing abilities [11], best approximation in integral sense [8, 10].

3 RBF ϕ Neural Networks

Besides fuzzy techniques, neural networks are also often used soft computing techniques (not only) for an approximation of functions. Compared to fuzzy techniques, they are usually implemented as black boxes but they have also advantages like e.g. algorithmic approach to an identification of a model or *on-line* learning algorithms. This section is devoted to the so called ϕ-neural nets which are studied e.g. in [7]. Basically, ϕ-neural nets are one hidden layer nets with only one linear unit (with an identity activation function) in the output layer.

Since the components F_i provide us with an information about the function f above its subdomains given by the supports of $\mathbf{A}_i$ they can be viewed as local units in the neural network terminology. Therefore the F-transform technique is closely related to the so called RBF (*Radial Basis Function*) neural networks which deal with the local units. Therefore in the latter we will consider only RBF ϕ-neural nets. Obviously, there exists a neural network performing the F-transform approximation i.e. the inverse F-transform, see Figure 1.

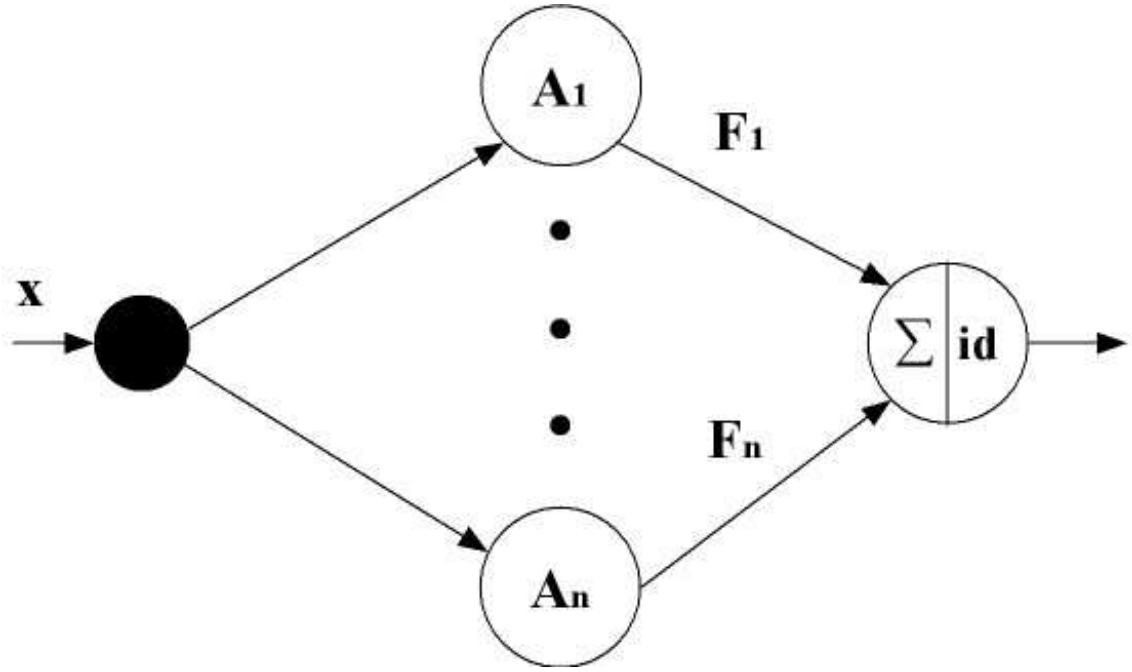

Figure 1: RBF ϕ-neural network performing the F-transform

There are different definitions and approaches to local unit activation functions or radial basis functions. In most cases, the Gaussian functions are used, see [5, 13].

The most usual approach to general RBF units is as follows: the activation function is basically a continuous non-increasing function $\mathbf{A} : \mathbb{R}^+ \to [0, 1]$ (compared to the perceptron neural nets where we require non-decreasing activation function, see [14]); the inner potential compared to the perceptron

neural networks is not computed as a weighted sum of inputs and weights but according to the following formula

$$\xi = \frac{||\mathbf{x} - \mathbf{c}||}{h} \tag{7}$$

where $\mathbf{x} \in \mathbb{R}^m$ is an input vector, $\mathbf{c} \in \mathbb{R}^m$ is a vector determining so called *center* of the unit and finally, $h \in \mathbb{R}^+$ is a parameter determining the width of the unit, see [14, 3]. In the latter, we restrict our focus to the case $m = 1$ for a simplified visualization.

Such a network can be constructed similarly to the one on Figure 1 with a few differences. First, all hidden layer units will provide the same activation function $\mathbf{A}$. Second, the input to the i-th hidden layer unit will be marked by the weight c_i determining the center of the unit. Third, each hidden layer unit will have a bias h_i determining width parameter of the unit. Fourth, the inner potential $\xi_i \in \mathbb{R}^+$ of the i-th unit is computed according to (8) i.e.

$$\xi_i = \frac{|x - c_i|}{h_i}, \tag{8}$$

see Figure 2.

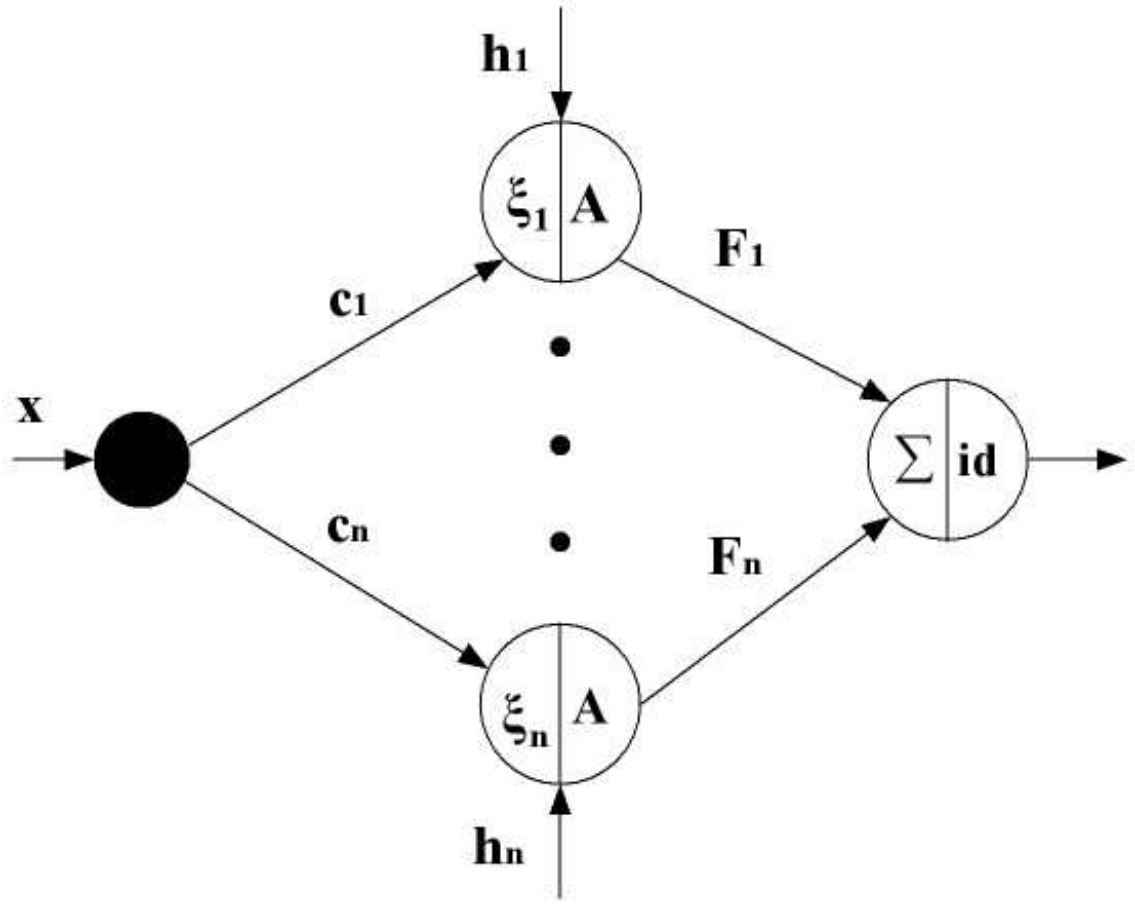

Figure 2: RBF ϕ-neural network

It is easy to see that the basic functions from Definition 1 can be constructed in the presented RBF neural network way. For instance, if we take $\mathbf{A}(\xi) = (1 - \xi) \wedge 0$ then it is easy to check that

$$\mathbf{A}_i(x) = \mathbf{A}(\xi_i) \tag{9}$$

where $\mathbf{A}_i$ are triangular shaped basic functions and $h = h_i$ for $i = 1, \ldots, n$. Similarly, if we take

$$\mathbf{A}(\xi) = \begin{cases} \frac{1}{2}\left(\cos\left(\Pi\xi\right) + 1\right) & \xi \leq 1, \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

which is since $\xi \in \mathbb{R}^+$ obviously a non-increasing function then one can again check that equality (9) holds for sinusoidal shaped basic functions $\mathbf{A}_i$.

4 Learning algorithm

Although both methods - RBF ϕ-neural networks and the F-transform - deal with an approximation of a function and both are, as discussed in the previous section, closely related to each other there are significant differences between them. May be the most important one is that neural network approach is in principle an algorithmic approach and can provide us with the so called *on-line* (or incremental) learning.

In the terminology of the neural nets, the computation of the components of the F-transform F_i according to (5) is called just an *off-line* (or batch) learning. However, for certain applications on-line learning algorithms have to be used.

The neural network approach can help us to keep all the nice properties of the F-transform and provide us with a technique belonging to on-line learning algorithms. So, from the original definitions we keep only the inverse F-transform formula which is performed by the RBF neural net displayed on Figure 1 and criterion (4) which is to be minimized. Formula (5) will be replaced by an on-line algorithm.

The most usual way how to construct an on-line learning is to consider it in the *delta rule* i.e. weights are modified by some delta after each new sample $(x_k, f(x_k))$ is involved. The gradient descent method is a standard tool for finding the delta.

To minimize the error function $E = \sum_{k=1}^{K} E_k$ given by (2) after each new sample we differentiate

$$\frac{\partial E_k}{\partial Q_i} \quad i = 1, \ldots, n. \tag{11}$$

Obviously,

$$\frac{\partial E_k}{\partial Q_i} = 2\mathbf{A}\left(\xi_i^{(k)}\right)\left(f(x_k) - Q_i\right) \tag{12}$$

where

$$\xi_i^{(k)} = \frac{|x_k - c_i|}{h}. \tag{13}$$

The gradient points at the direction of the fastest growth of the function values and therefore we will use the negative gradient in the construction of the delta rule. Therefore the delta rule is as follows

$$F_i^{(k)} = F_i^{(k-1)} + \theta_1(f(x_k) - F_i^{(k-1)})\mathbf{A}\left(\xi_i^{(k)}\right) \tag{14}$$

where $0 \leq \theta_1 \leq 1$ is a learning coefficient and $F_i^{(k)}$ is the i-th component of the F-transform after k samples involved where $k = 1, \ldots, K$.

Remark 1 *Notice, that although we use standard RBF neural network and standard neural tools like the gradient descent method together with the delta rule, the error function which is minimized is different compared to usual approaches. We do not compare function values $f(x_k)$ with the outputs of the network but with its weights $F_i^{(k-1)}$. This is a significant difference which is inherited from the F-transform to keep its properties.*

5 Learning of other parameters

In the previous section, we have introduced a relationship between the F-transform method and the RBF neural networks and inherited the gradient descent method for a learning algorithm. Besides the learning of the weights F_i we can get more from the neural network approach.

The construction of the basic functions can be the key issue for the results of the approximation. In general, one can hardly expect that the uniform distribution of the basic functions of the same length would provide us with the best results but on the other hand, the basic functions cannot be chosen arbitrarily and some say fuzzy cluster analysis would have to be used. Therefore, in most applications, the uniform fuzzy partition has been chosen. Let us discuss the possibility of the neural approach to the fuzzy partition construction.

Let us consider the nonsymmetric basic functions $\mathbf{A}_i$ then these basic functions are functions of four variables x, c_{i-1}, c_i, c_{i+1}. Therefore, in the latter, we will again use the notation from the F-transform since it is shorter i.e.

$$\mathbf{A}_i(x) = \mathbf{A}(x, c_{i-1}, c_i, c_{i+1}) \tag{15}$$

for $i = 1, \ldots, n$.

For instance, the nonsymmetric triangular shaped basic functions are given by

$$\mathbf{A}_i(x) = \begin{cases} \frac{(x-c_{i-1})}{c_i-c_{i-1}} & x \in [c_{i-1}, c_i] \\ \frac{(c_{i+1}-x)}{c_{i+1}-c_i} & x \in [c_i, c_{i+1}] \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

where $i = 0, \ldots, n+1$ and $c_0 = c_1,\ c_{n+1} = c_n$, while the nonsymmetric sinusoidal shaped basic functions are given by

$$\mathbf{A}_i(x) = \begin{cases} \frac{1}{2}\left(\cos\left(\frac{\Pi(x-c_i)}{(c_i-c_{i-1})}\right)+1\right) & x \in [c_{i-1}, c_i] \\ \frac{1}{2}\left(\cos\left(\frac{\Pi(x-c_i)}{(c_{i+1}-c_i)}\right)+1\right) & x \in [c_i, c_{i+1}] \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where $i = 0, \ldots, n+1$ and $c_0 = c_1,\ c_{n+1} = c_n$.

If we deal with a real-time problem which is necessary to be solved by an on-line learning algorithm we cannot simply use a fuzzy cluster analysis e.g. *fuzzy c-means*, see [1]. For these problems a neural approach again seems to be very appropriate since a lot of incremental self-organizing (unsupervised) algorithms have been already developed, see [5, 7]. We adopt a *simple c-means clustering* for RBF neural networks published e.g. in [14]. The task is to find the centroids c_i for $i = 1, \ldots, n$ which for a given shape of basic functions already completely specify the fuzzy partition.

The resulting algorithm using both, the self-organizing method for determining a distribution of the nodes c_i and the gradient descent method for adapting the components F_i will be as follows.

Algorithm:

(18)

FOR $k := 1$ `TO` K **DO BEGIN**
  $j = \mathtt{arg}\min_{i=1,\ldots,n}\{|x_k - c_i^{(k-1)}|\}$;
  FOR $i := 1$ **TO** n **DO BEGIN**
   IF $i = j$ **AND** $j \notin \{1, n\}$ **THEN**
   $c_i^{(k)} := c_i^{(k-1)} + \theta_2(x_k - c_i^{(k-1)})$
   ELSE
   $c_i^{(k)} := c_i^{(k-1)}$;
   $F_i^{(k)} = F_i^{(k-1)} + \theta_1(f(x_k) - F_i^{(k)})\mathbf{A}_i(x_k)$;
  END;
END.

The inputs $F_i^{(0)}$ for $i = 1, \ldots, n$ to the algorithm described above are small random numbers and $c_i^{(0)}$ for $i = 0, \ldots, n+1$ are distributed equidistantly on the domain and keeping the conditions $c_0 = c_1 = a$ and $c_n = c_{n+1} = b$.

The algorithm is independent on the shape of the basic functions. In its first part, it searches for the closest centroid to an actual incoming value x_k. The chosen centroid is then shifted unless it is a corner centroid c_1 or c_n. Then the delta rule formula is applied to each component F_i but because of the influence of the basic function $\mathbf{A}_i$ weighting the formula only two neighboring components F_i are modified at most.

6 Demonstration

Let us consider a function f given by

$$f(x) = 2e^{(-40(x-0.5))} - 1 \tag{19}$$

on a domain $[a, b] = [0, 1]$. Function (19) has been sampled to get a training set $(x_k, f(x_k))$ at randomly chosen nodes x_k where $k = 1, \ldots, K = 100$. For simplicity, we consider only one learning coefficient $\theta = \theta_1 = \theta_2$.

It is clear that if we implement the on-line learning given by (14) we never reach the accuracy obtained in case when the components are given by original formula (5). The components given by the delta rule only tend to the optimal ones given by (5).

Of course, better results were reached by resulting algorithm (18) which besides the components also modifies the distribution of the nodes c_i. It is not possible to measure the accuracy of the approximations by error function (4) since the error is weighted by the basic functions and the basic functions are different for both approximations.

Let us measure the error by *simple normed least square criterion* i.e. let

$$Error = 100\frac{1}{K}\sum_{k=1}^{K}\frac{(\hat{f}(x_k) - f(x_k))^2}{(\max f(x_k) - \min f(x_k))} \tag{20}$$

where $\hat{f}$ is the approximate output.

Results for the resulting neural algorithm were very often even better than results given by the original batch formula. For instance, for $n = 10$ the original approach gives results with 0.523 error for the triangular shaped basic functions (see Figure 3) and 0.462 for the sinusoidal shaped basic functions while the neural approach gives always different errors depending on random generation of $F_i^{(0)}$ and the choice of θ but sometimes even better than the original approach, see Table 1.

For the number of basic functions $n = 7$ the advantages of nonequidistant distribution of nodes c_i was naturally even stronger. For sinusoidal shaped basic functions the batch learning gives 1.227 error and for the triangular shaped basic functions error of 1.521, the neural approach experimental results are displayed in Table 2. On the other hand, the higher number of basic functions is the weaker advantage from the centroids modification we get and the on-line learning will hardly reach as good results as the batch one with fixed basic functions.

Remark 2 *Values in Tables 1 and 2 are experimental and only informative since the computations starts from randomly generated components* $F_i^{(0)}$.

7 Discussion

We have briefly recalled the F-transform technique as a robust fuzzy approximation method as well as the RBF neural networks as appropriate tolls for an approximation of functions. We discussed the

Table 1: Table of errors, $n = 10$.

θ	**Basic f. type**	**Error**
0.4	sin	1.427
0.4	tri	1.564
0.5	sin	0.977
0.5	tri	1.000
0.6	sin	0.708
0.6	tri	0.674
0.7	sin	0.534
0.7	tri	0.497
0.8	sin	0.467
0.8	tri	0.409
0.9	sin	0.457
0.9	tri	0.385
1.0	sin	0.485
1.0	tri	0.405

Table 2: Table of errors, $n = 7$.

θ	**Basic f. type**	**Error**
0.4	sin	0.985
0.4	tri	1.026
0.5	sin	0.775
0.5	tri	0.756
0.6	sin	0.676
0.6	tri	0.617
0.7	sin	0.616
0.7	tri	0.532
0.8	sin	0.587
0.8	tri	0.484
0.9	sin	0.578
0.9	tri	0.464
1.0	sin	0.601
1.0	tri	0.476

relationship between both approaches since the inverse F-transform mapping can be realized by an RBF neural network with basic functions as its activation function in its hidden layer. The difference in both methods is in learning of parameters especially the components F_i.

We have introduce a simple delta rule based on the gradient descent method determining the components by the so called on-line (incremental) learning what is an obvious advantage but on the other hand, such an algorithmic approach can never reach as good results as the original optimized batch formula.

The main benefit is that we can adapt also other approaches and techniques already developed in the neural network area. One of them, unsupervised c-means clustering for determining an appropriate distribution of nodes c_i for nonuniform fuzzy partition, was implemented to increase the approximation accuracy of the model. Obviously, better results would be obtained by off-line cluster analysis e.g. by fuzzy c-means and afterwards by applying the batch formula for the components. On-line algorithms can hardly compete with them. On the other hand, from the computational complexity point of view it does not have to be always efficient and therefore usually only uniform distributions were used. Moreover, there are such real-time application where such clustering in advance can be impossible. Let us mention at least one of them aiming at real time prediction of emission of a diesel engine which was studied in [4].

Undoubtable disadvantage of the introduced approach is, that we loose linearity property (6) if we

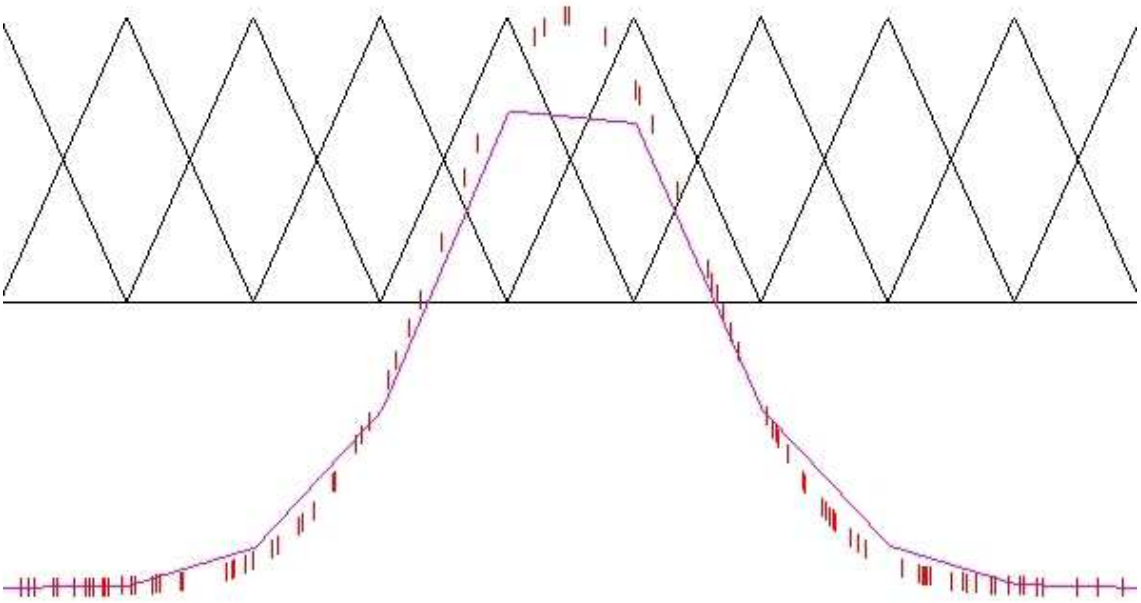

Figure 3: Approximation of sampled function (19) by the original F-transform; $N = 10$ triangular shaped basic functions.

adapt the centroids since they are adapted for each function in a different way. It should be stressed that such approach is appropriate only if we want to approximate a function or some dependencies between variables not if we intend to transform several functions into a discrete space, deal with their component vectors and then transform them back.

From the experimental part of the paper, we find the results to be promising for future. The paper is considered to be an introduction to this new stage of investigation which promises to inherit results from the neural network area to the fuzzy approximation area and viceversa. For instance, how to deal with the parameters θ_1 and θ_2 if they vary in time is a natural question.

References

[1] J.C. Bezdek (1981). *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York.

[2] J. Buckley and Y. Hayashi (1993). Fuzzy Input-Output Controllers Are Universal Approximators. *Fuzzy Sets and Systems*, volume 58, pages 273-278.

[3] D. Coufal (2005). Radial Implicative Fuzzy Systems. *Proceedings of the FUZZ-IEEE2005 conference*, pages 963-968, Reno, Nevada, May 2005.

[4] E. Lughofer (2005). *Data-Driven Incremental Learning of Takagi-Sugeno Fuzzy Models*, PhD-Thesis, Department of Knowledge-Based Mathematical Systems, University Linz, Austria.

[5] R. Fullér (2000). *Introduction to Neuro-Fuzzy Systems (Advances in Soft Computing Series)*, Springer-Verlag, Berlin, Heidelberg.

[6] B. Kosko (1992). Fuzzy Systems as Universal Approximators. In *Proceedings of the FUZZ-IEEE1992 conference*, San-Diego, California, pages 1153-1162.

[7] V. Kurková (2003). Approximation of Functions by Neural Networks. In *V. Mařík, O. Štěpánková and J. Lažanský (Eds.) Artificial intelligence*, volume 4, pages 254-273, Academia, Prague, in Czech.

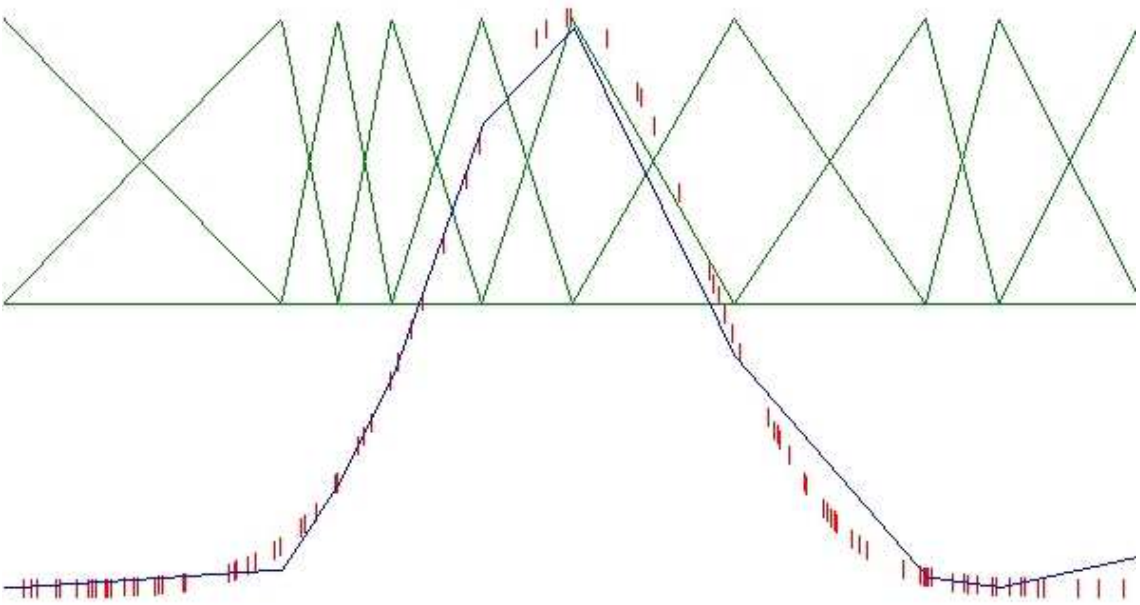

Figure 4: Approximation of sampled function (19) by the neural F-transform; $N = 10$ triangular shaped basic functions; $\theta_1 = \theta_2 = 0.8$.

[8] I. Perfilieva (2003). Fuzzy Approach to Solution of Differential Equations with Imprecise Data: Application to Reef Growth Problem. In *R.V. Demicco and G. J. Klir (Eds.) Fuzzy logic in geology*, pages 275-300, Academic Press, Amsterdam.

[9] I. Perfilieva and E. Haldeeva (2001). Fuzzy Transformation and Its Applications. In *Proceedings of the 4th Czech - Japan seminar on data analysis and decision making under uncertainity*, pages 116-124.

[10] I. Perfilieva (2006). Fuzzy Transforms. *Fuzzy Sets and Systems*, to appear.

[11] I. Perfilieva and R. Valášek (2005). Fuzzy Transforms in Removing Noise. In *B. Reusch (Eds.) Computational Intelligence, Theory and Applications (Advances in Soft Computing)*, pages 225-236, Springer-Verlag, Berlin.

[12] E.H. Ruspini (1969). A New Approach to Clustering. *Inform. and Control*, volume 15, pages 22-32.

[13] D. Rutkowska (2002). *Neuro-fuzzy Architectures and Hybrid Learninng*, Springer-Verlag, Heidelberg, New York.

[14] J. Šíma and R. Neruda (1996). *Theoretical Problems of Neural Networks*. Matfyzpres, Prague, in Czech.

[15] M. Štěpnička R. Valášek (2004). Fuzzy Transforms and Their Application to Wave Equation. *Journal of electrical engineering*, volume 55, number 12/s, pages 7-10.

[16] M. Štěpnička R. Valášek (2005). Numerical Solution of Partial Differential Equations with Help of Fuzzy Transform. In *Proceedings of the FUZZ-IEEE2005 conference*, pages 1104-1109, Reno, Nevada, May 2005.