International Journal of Computational Intelligence Research.
ISSN 0973-1873 Vol.1, No.1 (2005), pp. 29-41
© Research India Publications http://www.ijcir.info

Approximation Capabilities of Hierarchical Neural-Fuzzy Systems for Function Approximation on Discrete Spaces

Xiao-Jun Zeng
School of Informatics
University of Manchester, U.K.
e-mail: x.zeng@manchester.ac.uk

John A Keane
School of Informatics
University of Manchester, U.K.
e-mail: john.keane@manchester.ac.uk

John Yannis Goulermas
Department of Electrical Engineering and Electronics
University of Liverpool, U.K.
e-mail: j.y.goulermas@liverpool.ac.uk

Panos Liatsis
Information and Biomedical Engineering Centre
School of Engineering and Mathematical Sciences
City University, London EC1V 0HB, U.K.
e-mail: p.liatsis@city.ac.uk

***Abstract*—This paper investigates function approximation on discrete input spaces by both neural networks and neural-fuzzy systems. Rather than use existing neural networks for function approximation on continuous input spaces, this paper proposes, based on a hierarchical systematic perspective, four simplified approximation schemes: *simplified neural networks, extended simplified neural networks, simple hierarchical neural-fuzzy systems* and *hierarchical neural-fuzzy systems*. Each scheme is proven to be a universal approximator (i.e., each can approximate any function on discrete input spaces to any degree of accuracy). The results provide both several new and simpler approximation schemes for function approximation on discrete spaces and show that there exist simpler and more effective methods for function approximation on discrete spaces compared with continuous spaces.**

Keywords: Neural Networks, Fuzzy Systems, Neural-Fuzzy Systems, Hierarchical systems.

I. Introduction

Approximation or representation capabilities of neural networks and fuzzy systems have attracted considerable research in the last 15 years. In neural networks, following from the proof of their universal approximation property (Cybenko [5], Hecht-Nielsen[11], and Carroll and Dickinson [3], Hornik, Stinchcombe, and White[12]), it has been proved that various neural networks are universal approximators and that the various results on approximation accuracy are also available (see, for example, [1], [13]-[15], [17], [18], [20], [21], and [23]). In fuzzy systems, the work on their approximation capabilities (Buckley [2], Kosko[16] and Wang [24]) has shown that fuzzy systems are also universal approximators. Since then, a number of results related to approximation capabilities and accuracy have been published (see, for example, [4], [10], [19], [28], and [29]-[31]); more recently these results have been extended to hierarchical and hybrid systems (see, for example, [8], [22], [25], [26], [33] and [34]). In addition to research on neural networks' and fuzzy systems' approximation capability, the approximation capabilities of wavelets and support vector machines (SVM) have been investigated (see, for example, [6] and [9]). However, almost all these available results focus on function approximation on continuous input spaces with few results available for function approximation on discrete spaces. This may be because function approximation on discrete input spaces can be viewed as a special case of function approximation on continuous spaces as any function on discrete spaces can be expanded to be a continuous function [27] that interpolates the given discrete function, and then the existing results in function approximation on continuous spaces imply that neural networks and fuzzy systems are universal approximators for functions defined on discrete spaces. Although such a view is both valid and correct, it ignores the difference between function approximation on continuous and discrete spaces, especially the potential to develop simpler approximation schemes based on neural networks and fuzzy systems for function approximation on discrete input spaces.

In this paper, motivated by this potential, the approximation capabilities of neural networks and neural-fuzzy systems for function approximation on discrete spaces are investigated by focusing on the distinguishing features of discrete input spaces. Several new simplified approximation schemes designed specially for function approximation on discrete spaces are proposed:

1. *Simplified Neural Networks (SNNs)*

2. *Extended Simplified Neural Networks (ESNNs)*
3. *Simple Hierarchical Neural-Fuzzy Systems (SHNFSs)*
4. *Hierarchical Neural-Fuzzy Ssystems (HNFSs)*

The universal approximation property (i.e., the capability to approximate any function on discrete input spaces to any degree of accuracy) of these approximation schemes are then proved. In other words, for function approximation on discrete input spaces, the proposed approximation schemes are simpler and more effective whilst remaining as general as those approximation schemes in the literature for neural networks and fuzzy systems.

The paper is structured as follows: Section II proposes the four simplified approximation schemes for function approximation on discrete input spaces and analyzes their utility; Section III analyzes the approximation capabilities of the proposed approximation schemes and presents their universal approximation properties; finally conclusions are presented in Section IV, and the proofs of all theorems presented are given in the Appendix.

II. Simplified Neural Networks and Hierarchical Neural-Fuzzy Systems

Throughout the rest of the paper, it is assumed that the system or function to be modelled or approximated is a multi-input single-output (MISO) function defined on a discrete space. That is, suppose that the function is given as follows:

$$y = G(X) = G(x_1, x_2, ..., x_n) \tag{1}$$

where $y \in V \subset R$ is the output variable and $X = (x_1, x_2, ..., x_n) \in U = U_1 \times U_2 \times ... \times U_n \subset R^n$ is the input variable vector in which $x_i \in U_i$ and

$$U_i = \{u_{i,k} \mid u_{i,k} \in R, k = 1,2,..., N_i\} \tag{2}$$

In other words, input variable x_i takes discrete values.

In the following, simplified (feedforward) neural networks and hierarchical neural-fuzzy systems are proposed to approximate functions on discrete spaces, i.e., those functions given in (1) and (2).

A. Simplified Neural Networks (SNNs)

The standard and most commonly used (feedforward) neural networks (NN) can be represented as:

$$y = NN(X) = \sum_{i=1}^{N} c_i \sigma(a_i^\tau X + b_i) + c_0 \tag{3}$$

where $X = (x_1, x_2, ..., x_n)$ are input variables, $X \in U = U_1 \times U_2 \times ... \times U_n \subset R^n$ which are input space, $y \in R$ is the output variable, τ is the vector transpose, $\sigma(.)$ is the activation function and the parameters $c_0 \in R$, $c_i \in R$, $a_i \in R^n$, and $b_i \in R$ $(i = 1,2,..., N)$.

Given the standard NN given in (3), the total number of parameters [i.e., $c_i \in R$, $a_i \in R^n$, $b_i \in R$ ($i = 1,2,..., N$) and $c_0 \in R$] is $(n+2)N + 1$. For nonlinear complex function approximation, a large *N* is needed and often N will grow exponentially with the dimension of n [1]. As a result, a large number of parameters are needed in order to achieve good approximation accuracy.

To overcome this difficulty, *a simplified neural network (SNN)* is proposed for function approximation on discrete spaces as follows:

$$y = SNN(X) = \sum_{i=1}^{N} c_i \sigma[\alpha_i(\alpha^\tau X + \beta) + \beta_i] + c_0 \tag{4}$$

where $c_0 \in R$, $c_i \in R$, $\alpha_i \in R$, $\beta_i \in R$ ($i = 1,2,..., N$) and $\alpha \in R^n$, $\beta \in R$.

Let

$$z = L(X) = \alpha^\tau X + \beta \tag{5}$$

and

$$y = NN_1(z) = \sum_{i=1}^{N} c_i \sigma(\alpha_i z + \beta_i) + c_0 \tag{6}$$

Then the proposed SFNN given in (4) can be rewritten as follows:

$$y = SNN(X) = NN_1[L(X)] \tag{7}$$

In other words, the proposed SNN can be presented as a composition of a linear function $L(X)$ given in (5) and a one-dimensional standard NN $NN_1(z)$ given in (6).

For the SNN given in (4), the total number of parameters is $3N + n + 2$. Therefore, in many cases fewer parameters are needed for SNNs in comparison to the number needed for standard NNs. Another advantage of SNNs is that they are more effective in overcoming the model over-fitting which often happens in NN modeling. This is because: in the standard NNs, adding a new neuron [i.e., add an item $c_i \sigma(a_i^\tau X + b_i)$ in(3)] means adding $n+2$ parameters. As a result, in NN modeling it often happens that adding one more neuron causes model overfitting whereas not adding such a new neuron may result in underfitting, especially in the case where n is large but only limited training data is available. However, in SNNs, adding a new neuron means adding an item $c_i \sigma(\alpha_i z + \beta_i)$ which only adds three parameters. As a result, SNNs allow the addition of finer-grained parameters to overcome model overfiting and underfitting, especially in the high dimension (i.e., large n) case. Another potential advantage is that simpler learning algorithms can be developed. For example, in some cases multi-dimensional NN learning problems can be transformed to a one-dimensional NN learning problem and thus the corresponding learning algorithms can be much simpler (see Section III for more detailed discussion on this).

To approximate a function $G(X)$ given in (1) on discrete space $U=\prod_{i=1}^{n}U_i$ given in (2), the basic idea in using the SNNs is that, firstly a linear function $L(x)$ is constructed to transform n dimensional variables $X=(x_1,x_2,...,x_n)$ into a one-dimensional variable z and then a one-dimensional standard NN $NN_1(z)$ is constructed to form the final SNN $SNN(X)=NN_1[L(X)]$ to approximate the given function $G(X)$. A major focus of this paper is to prove that SNNs have the same universal approximation property (i.e., they are able to approximate any function to any degree of accuracy) as standard NNs, that is, to prove the feasibility and general applicability of SNNs as a new and simpler NNs for function approximation on discrete spaces.

There are two possible views on the SNN given in (4). Firstly, it can be viewed as a special case of the standard three layered feedforward NN given in (3) in which the parameters take the particular form of $a_i=\alpha_i\alpha$ and $b_i=\alpha_i\beta_i+\beta_i \quad (i=1,2,...,N)$. Secondly, it can be viewed as a hierarchical hybrid NN system in which the lower level sub-system is a linear function given in (5) and the higher level sub-system is a one-dimensional NN given in (6) which takes the output variable of the lower sub-system as its input variable. The combination of the two sub-systems forms the hierarchical hybrid system given in (7) which is the proposed SNN. Although both views produce the same SNNs given in (4) in this instance, the second view is more general and flexible. The extended SNNs and the hierarchical neural-fuzzy systems proposed later in the paper result from this view. It should be noted that hierarchical neural-fuzzy systems can only be obtained from the second hierarchical hybrid systems view as they are no longer a special case of standard NNs.

An *extended SNN (ESNN)* differs from a SNN in that, rather than using one linear function to transform n dimensional variables $X=(x_1,x_2,...,x_n)$ into a one-dimensional variable z, it uses $m(<n)$ sub-linear functions as the lower level sub-systems to transform n dimensional variables $X=(x_1,x_2,...,x_n)$ into m dimensional variables $Z=(z_1,z_2,...,z_m)$ and then use a m dimensional standard NN (which takes the output variables of the lower sub-systems as its input variables) as the higher level sub-system. The detailed mathematical formula of an ESNN is as follows:

Let $G_j \ (j=1,2,...,m)$ be a disjoint grouping of the input variables $\{x_1,x_2,...,x_n\}$ as follows:

$$G_j=\left\{x_{i_1^{(j)}},x_{i_2^{(j)}},...,x_{i_{n_j}^{(j)}}\right\} \qquad j=1,2,...,m \tag{8}$$

where

$$G_j\cap G_{j'}=\varnothing \qquad j\neq j', \quad j,\ j'=1,2,...,m \tag{9}$$

$$G_1\cup G_2\cup....\cup G_m=\{x_1,x_2,...,x_n\} \tag{10}$$

and $\sum_{j=1}^{m}n_j=n$. Let $X_j=(x_{i_1^{(j)}},x_{i_2^{(j)}},...,x_{i_{n_j}^{(j)}})$ denote the input variables of group $G_j \ (j=1,2,...,m)$, and then the lower level sub-systems are linear functions given by

$$z_j=L_j(X_j)=\phi_j^{\tau}X_j+\varphi_j \qquad j=1,2,...,m \tag{11}$$

where $\phi_j\in R^{n_j}$, $\varphi_j\in R$ and $X_j\in U_{G_j}=\mathop{\times}_{k=1}^{n_j}U_{i_k}\subset R^{n_j}$. Further the higher level sub-system is a m dimensional standard NN which takes the output variables of the lower level sub-systems as its input variables and is given by

$$\begin{aligned} y&=NN_m(Z)=NN_m(z_1,...,z_m)\\ &=\sum_{i=1}^{N}c_i\sigma(\alpha_i^{\tau}Z+\beta_i)+c_0 \end{aligned} \tag{12}$$

with $c_0\in R$, $c_i\in R$, $\alpha_i=[\alpha_{i1},\alpha_{i2},...,\alpha_{im}]^{\tau}\in R^m$, $\beta_i\in R \quad (i=1,2,...,N)$. Finally the ESNN is the following hierarchical system formed by combining the above sub-systems as

$$\begin{aligned} y&=ESNN(X)=NN_m[L_1(X_1),...,L_m(X_m)]\\ &=\sum_{i=1}^{N}c_i\sigma\left[\sum_{j=1}^{m}\alpha_{ij}L_j(X_j)+\beta_i\right]+c_0\\ &=\sum_{i=1}^{N}c_i\sigma\left[\sum_{j=1}^{m}\alpha_{ij}(\phi_j^{\tau}X_j+\varphi_j)+\beta_i\right]+c_0 \end{aligned} \tag{13}$$

where the parameters $c_0\in R$, $c_i\in R$, $\beta_i\in R$, $\alpha_i=[\alpha_{i1},\alpha_{i2},...,\alpha_{im}]^{\tau}\in R^m$, $i=1,2,...,N$, and $\phi_j\in R^{n_j}$, $\varphi_j\in R$, $j=1,2,...,m$. As $\sum_{j=1}^{m}n_j=n$, then the total number of parameters of the ESNN is $1+(m+2)N+\sum_{j=1}^{m}n_j+m=(m+2)N+n+m+1$. As $m<n$, therefore, the ESNN can use fewer parameters in function approximation.

On the one hand, the ESNN given above can be viewed as a special case of the standard NN in which $a_i=[\alpha_{i1}\phi_1,...,\alpha_{im}\phi_m]$ and $b_i=\sum_{j=1}^{m}\alpha_{ij}\varphi_j+\beta_i$ ($i=1,2,...,n$). On the other hand, the SNN given in (4) can be viewed as the special case of the ESNN when m=1, and the standard NN given in (3) can be viewed as the special case of the ESNN when $m = n$ and $\phi_j = 1, \varphi_j = 0$ ($j = 1,2,...,n$). In other words, ESNNs are very flexible with regard to model complexity, lying somewhere between SNNs and standard NNs respectively.

From an application viewpoint, the main reason to introduce ESNNs is their flexibility as the number of input variable groups and the input variables in each group can be chosen based on the need and desire of each application. For example, in applications of high dimensional complicated system modeling, it is often desired to classify the large number of input variables into different groups and then identify the impact of each input variable group on the system output. ESNNs can achieve this by using each lower level sub-system to transform each group of input variable into its single output variable into the higher level sub-system and the impact of each group to the system output can be seen by the corresponding input-output relationship at the higher level sub-system. In addition, the representation accuracy of float numbers may make SNNs difficult to use in some high dimensional cases and then ESNNs are needed (see Section III for more detailed discussion about this point).

B. *Hierarchical Neural-Fuzzy Systems (HNFSs)*

Taking the hierarchical hybrid view of SNNs mentioned in the last subsection by replacing the linear function $L(X)$ by a fuzzy system $F(X)$, a *Simple Hierarchical Neural-Fuzzy system (SHNFS)* can be obtained as follows:

The lower level sub-system is a fuzzy system $F(X)$ whose rule base is given as:

R^l: IF x_1 *is* $A_{1,l}$ and … and x_n *is* $A_{n,l}$,

THEN z *is* C_l $\quad l = 1,2,...,L$ (14)

and its mathematical formula is represented by

$$z = F(X) = \sum_{l=1}^{L} B_l(X) y_l \tag{15}$$

where y_l is the centroid of the output fuzzy set C_l,

$$B_l(X) = \frac{A_l(X)}{\sum_{l=1}^{L} A_l(X)}$$

are fuzzy basis functions [32] (also called normalized membership functions [7]) and $A_l(X) = \prod_{i=1}^{n} A_{i,l}(x_i)$ are the membership functions $(l = 1,2,...,L)$.

The higher level sub-system is a one-dimensional standard NN given in (6) and then the final SHNFS is given by

$$\begin{aligned} y = SHNFS(X) &= NN_1[F(X)] \\ &= \sum_{i=1}^{N} c_i \sigma[\alpha_i F(X) + \beta_i] + c_0 \\ &= \sum_{i=1}^{N} c_i \sigma\left\{\alpha_i \left[\sum_{l=1}^{L} B_l(X) y_l\right] + \beta_i\right\} + c_0 \end{aligned} \tag{16}$$

where $c_0 \in R$, $c_i \in R$, $\alpha_i \in R$, $\beta_i \in R$ $(i = 1,2,...,N)$.

Compared with SNNs introduced above, SHNFSs have several features which could be useful in applications. Firstly, as the lower level sub-system is a nonlinear fuzzy system, such SHNFSs have better representation power whilst still being relatively simple and transparent due to the rule representation and interpretability of fuzzy systems. This improved representation power in the lower level allows the higher level NN sub-system to be simpler which can lead to fewer parameters and less training data being needed in the higher level NN sub-system modeling. Secondly, it enables the combination of human (knowledge and experience) and machine intelligence (learning from data) in system modeling. That is, the fuzzy systems method can utilize human intelligence to form the lower level fuzzy sub-system and then the learning algorithms of neural networks can be applied to identify the higher level NN model from the available numerical training data. This is very useful in applications where there is only limited training data but relevant human knowledge is available.

As with the ESNN discussion, in applications of high dimensional complicated system modeling, it is often desired to classify the large number of input variables into different groups and then identify the impact of each input variable group on the system output. In addition, in the high dimensional situation, utilizing human knowledge by one single fuzzy system is often infeasible as it will result in a few thousands or more rules to collect human knowledge. For example, for the simplest fuzzy systems in which each input variable has only two possible fuzzy values, the total number of rules is 2^n when there are n input variables. As a result, for high dimensional function approximation or system modeling, a more feasible and flexible hierarchical structure is needed. The following general *Hierarchical Neural-Fuzzy Systems (HNFS)* is proposed to meet these requirements.

Firstly, divide the input variables $\{x_1, x_2, ..., x_n\}$ into m disjoint groups G_j $(j = 1,2,...,m)$ as given in (8)–(10) and let $X_j = (x_{i_1^{(j)}}, x_{i_2^{(j)}}, ..., x_{i_{n_j}^{(j)}})$ denote the input variables of group G_j $(j = 1,2,...,m)$. Then the lower level sub-systems of a HNFS are fuzzy systems $F_j(X_j)$ $(j = 1,2,...,m)$ whose rule base is given as:

R_l^j: IF $x_{i_1^{(j)}}$ *is* $A_{1,l}^{(j)}$ and … and $x_{i_{n_j}^{(j)}}$ *is* $A_{n_j,l}^{(j)}$,

THEN z *is* $C_l^{(j)}$ $\quad l=1,2,...,L_j$ (17)

and its mathematical formula is represented by

$$z_j = F_j(X_j) = \sum_{l=1}^{L_j} B_l^{(j)}(X_j) y_l^{(j)} \tag{18}$$

where $y_l^{(j)}$ is the centroid of the output fuzzy set $C_l^{(j)}$,

$$B_l^{(j)}(X_j) = \frac{A_l^{(j)}(X_j)}{\sum_{l=1}^{L} A_l^{(j)}(X_j)}$$

are fuzzy basis functions, and $A_l^{(j)}(X_j) = \prod_{k=1}^{n} A_{k,l}^{(j)}(x_{i_k^{(j)}})$ are the membership functions $(l=1,2,...,L_j)$.

The higher level sub-system is a m dimensional standard NN given in (12) and then the final HNFS is given by:

$$y = HNFS(X) = NN_m[F(X)]$$

$$= \sum_{i=1}^{N} c_i \sigma[\alpha_i^{\tau} F(X) + \beta_i] + c_0 \tag{19}$$

$$= \sum_{i=1}^{N} c_i \sigma \left\{ \sum_{j=1}^{m} \alpha_{ij} \left[\sum_{l=1}^{L_j} B_l^{(j)}(X) y_l^{(j)} \right] + \beta_i \right\} + c_0$$

where $F(X) = [F_1(X),..., F_m(X)]^{\tau}$, $c_0 \in R$, $c_i \in R$, $\beta_i \in R$, $\alpha_i = [\alpha_{i1}, \alpha_{i2},...,\alpha_{im}]^{\tau} \in R^m$, $i=1,2,...,N$.

III. Approximation capabilities of SNNs and HNFSs

In this section, the approximation capability of SNNs is analyzed first. As has been explained, SNNs require fewer parameters for function approximation than standard NNs. However, an important question is whether such SNNs are general enough to approximate any function on discrete spaces, that is, whether SNNs preserve the universal approximation capabilities of standard NNs. The approximation capability analysis presented in this section provides a positive answer to this question.

In order to analyze the approximation capabilities of SNNs, a theorem is introduced first.

Theorem 1. Let $U = U_1 \times U_2 \times ... \times U_n$ be a discrete space in which $U_i = \{u_{i,k} \mid u_{i,k} \in R, k=1,2,..., N_i\}$ $(i=1,2,...,n)$.Then there exists a real value linear function $y = L(X) = L(x_1, x_2,..., x_n)$ defined on U such that $L: U \to R$ is a one-to-one mapping [i.e., if $X \neq X'$, then $L(X) \neq L(X')$].

Proof of this theorem first appeared in [35]. As this theorem is fundamentally important to the later analysis here, it is also included in the Appendix.

The above theorem shows that, for a discrete space $U \subset R^n (n \geq 2)$ as given in (2), there exist some simple functions such as linear functions which form one-to-one mappings from U to R. This is a property which holds only on discrete spaces but not on continuous spaces. This is because no one-to-one mapping from a multi-dimensional continuous space $U = \times_{i=1}^{n} [\alpha_i, \beta_i]$ $(n \geq 2)$ to R can be continuous [35]. As no continuous function can be found to form a one-to-one mapping from a multi-dimensional continuous space to R, it is impossible to find a simple function which is a one-to-one mapping from a multi-dimensional continuous space U to R. In other words, multi-dimensional information on discrete spaces can be coded into one dimension by using simple functions such as linear functions without loss of information but this cannot be achieved on continuous space. This is the main reason why function approximation on discrete spaces can be achieved by simpler approximation schemes than for continuous spaces, and it forms the basis for the results in this paper.

Based on Theorem 1, to approximate a multi-dimensional function $G(X)$ given in (1) on a discrete space U given in (2) can be done by two steps: firstly, use a simple one-to-one mapping $z = M(X)$ such as a linear function to transform the multi-dimensional discrete input space U into a one dimension discrete space V. As $z = M(X)$ is a one-to-one mapping from U to V, then its inverse function $X = M^{-1}(z)$ exists (notice here $M^{-1}(.)$ is a vector value function rather than a normal real value function). Then $G(X)$ can be represented as

$$G(X) = G[M^{-1}(z)]$$

As $g(z) = G[M^{-1}(z)]$ is a one-dimensional function on a discrete space V, then the original multi-dimensional function approximation problem becomes a one-dimensional approximation problem and a one-dimensional standard NN can be used to approximate $g(z)$ to achieve any desired approximation accuracy due to the universal approximation property of NNs. The following universal approximation theorem for SNNs is obtained based on this idea, with the detailed proof of the theorem given in the Appendix.

Theorem 2 (Universal Approximation Property of SNNs). Let $G(X)$ be a function on a discrete space

$U = U_1 \times U_2 \times ... \times U_n$ in which $U_i = \{u_{i,k} \mid u_{i,k} \in R, k = 1,2,..., N_i\}$ $(i = 1,2,...,n)$. Then for any given $\varepsilon > 0$, there exists a SNN $SNN(X)$ given in (4) such that

$$\| G - SNN \|_\infty = \max_{X \in U} | G(X) - SNN(X) | < \varepsilon \qquad (20)$$

Theorem 2 shows that SNNs can approximate any function on a discrete space to any degree of accuracy. In other words, SNNs, despite their simplified formula, preserve the universal approximation property of standard NNs and therefore are generally applicable for function approximation on discrete spaces. In the following, suppose that the available training data are given as $\{(X_t, y_t) \mid t = 1,2,..., N\}$, then two possible algorithms to find a SNN approximator for a given function are briefly discussed:

The first algorithm is based on the proof of Theorem 2 which includes two steps: the first step is to find a one-to-one linear mapping $L(X)$ from U to R and then a one-dimensional function $g(z) = G[L^{-1}(z)]$ or $g[L(X)] = G(X)$ can be defined; the second step is to use the available data $\{(X_t, y_t) \mid t = 1,2,..., N\}$ to get a set of training data for function $g(z)$ as $\{(z_t, y_t) \mid z_t = L(X_t), t = 1,2,..., N\}$ and then, for $g(z)$, apply the learning algorithms of standard NN to find a one-dimensional NN approximator $NN_1(z)$ with the required approximation accuracy. Finally, the SNN approximator can be obtained by $SNN(X) = NN_1[L(X)]$. Theoretically speaking, this is a very simple method as by using the one-to-one linear mapping $L(X)$, the original approximation problem is transformed to a simple learning problem of a single variable NN. In the case where the number of input variables and the possible values of each input variables are small, then this is a good algorithm in practice due to its simplicity. However, this method is not suitable for high dimensions (i.e., many input variables or n is large) with each input variable having many possible values (i.e., N_j is large). The reason is as follows: as the total number of all possible values of input vector $X = (x_1, x_2,..., x_n)$ are $\prod_{i=1}^{n} N_i$, the total number of the possible function values of a one-to-one mapping $z = L(X)$ is $\prod_{i=1}^{n} N_i$. When n and N_i $(i = 1,2,..., n)$ are large, this is impossible as all possible values are beyond the representation accuracy of float numbers. Therefore, in the case when n and N_i $(i = 1,2,..., n)$ are large, the implementation of this algorithm, as explained in the last section, requires use of ESNNs. More details about how to use ESNNs to handle such a situation are discussed later.

The second algorithm is to apply the gradient descent optimisation algorithms to minimise

$$E = \frac{1}{2} \sum_{t=1}^{T} [y_t - SNN(X_t)]^2$$

where $SNN(X)$ is given in (4) with the parameters $\{c_i, \alpha_i, \beta_i, \alpha, \beta, c_0 \mid i = 1,2,..., N\}$ to be identified. In this algorithm, it is not required that $z = L(X) = \alpha^\tau X + \beta$ is a one-to-one mapping (note that a one-to-one mapping is a sufficient but not a necessary condition), rather parameters α and β are tuned by the learning algorithm to meet the approximation requirement. This algorithm is more complicated than the first one but should be able to handle the higher dimensional modeling situation. In order to realize the potential of SNNs and apply them to applications, implementation and comparison of these two methods is needed.

The above discussion illustrates that the proposed SNN approximation scheme is realizable and applicable. However, as the main focus of this paper is the analysis of approximation capabilities rather than the development of algorithms to implement the proposed SNN approximation scheme, algorithm development is not discussed further.

The next step is to investigate the approximation capability of ESNNs. Similar to the earlier analysis of SNNs, the basic idea is as follows:

Based on Theorem 1, approximation of a n dimensional function $G(X)$ in a discrete space U by an ESNN can be achieved by two steps: firstly, use of several one-to-one mappings $z_j = M(X_j)$ $(j = 1,2,..., m)$ such as one-to-one linear functions to transform the n dimensional discrete input space $U = \times_{i=1}^{n} U_i = \times_{j=1}^{m} U_{G_j}$ [where $U_{G_j} = \times_{k=1}^{n_j} U_{i_k}$ ($j = 1,2,..., m$)] into m dimension discrete space $V = \times_{j=1}^{m} V_j$. That is, each $z_j = M(X_j)$ is a one-to-one mapping from U_{G_j} to V_j $(j = 1,2,..., m)$. Then $G(X)$ can be represented as

$$G(X) = G[M_1^{-1}(z_1),..., M_m^{-1}(z_m)]$$

As $g(Z) = g(z_1,..., z_m) = G[M_1^{-1}(z_1),..., M_m^{-1}(z_m)]$ is a m dimensional function on a discrete space V, then a m dimensional standard NN can be used to approximate $g(Z)$ to achieve any desired approximation accuracy. Based on such an idea, the following theorem about the approximation capability of ESNNs can be obtained.

Theorem 3 (Universal Approximation Property of ESNNs). Let $G(X)$ be a function on a discrete space $U = U_1 \times U_2 \times ... \times U_n$ in which $U_i = \{u_{i,k} \mid u_{i,k} \in R, k = 1,2,..., N_i\}$ $(i = 1,2,..., n)$. Then for any given $\varepsilon > 0$ and for any disjoint grouping of the input variables $\{x_1, x_2,..., x_n\}$ into m groups G_j $(j = 1,2,..., m)$ satisfying (8)–(10), there exists an ESNN $ESNN(X)$ given in (13) such that

$$\| G - ESNN \|_\infty = \max_{X \in U} | G(X) - ESNN(X) | < \varepsilon \quad (21)$$

The main advantage of the above theorem is that, for any disjoint grouping of the input variables $\{x_1, x_2,..., x_n\}$ (i.e., the user can choose the number of groups and which input variables are in which group), an ESNN with such an input variable grouping can be found to approximate the given function to any degree of accuracy. This is a useful property in applications as it means that an ESNN can be designed based on the required different impact of different groups of input variables on the system output. In other words, the ESNN both allows the required approximation accuracy and enables better understanding of system behavior.

The two possible algorithms proposed for SNNs are also applicable here. The only differences are as follows: in the first algorithm, m one-to-one linear mappings are needed from the sub-input-spaces U_{G_j} to V_j $(j = 1,2,..., m)$ rather than only one one-to-one linear mapping needed, and the higher level sub-system to be trained is a m dimensional NN rather than a one-dimensional NN. For function approximation in high dimensional input spaces, the whole input space can be divided into several disjoint sub-spaces such that a one-to-one linear mapping on each sub-space is possible within the representation accuracy of float numbers. In other words, high dimensional function approximation and modeling can be handled by proper ESNNs. Although the learning of the higher level sub-system is a more complicated m dimensional NN, it can still be much simpler than training a standard NN with n dimensions. Consider an example where $n = 25$. Assume we design 5 one-to-one linear functions in which each linear function takes 5 variables (today's computers are likely to be able to represent a 5-dimension one-to-one mapping), then the training of a 25-dimension standard NN in the existing NN learning methods can be transformed into the training of a 5-dimension NN by using the proposed ESNN method. In other words, ESNNs can handle the high dimensional modeling problem and can be much simpler than standard NNs in many cases.

Now the above results of SNNs and ESNNs are extended to SHNFSs and HNFSs. Such an extension is possible because fuzzy systems can realize any linear and many nonlinear functions [32]. That is, by choosing the commonly used triangle membership functions and proper system parameters, fuzzy systems can exactly represent any linear function. Based on Theorem 1, that there are one-to-one linear mappings from a multi-dimensional discrete space to a one-dimensional discrete space, it can be implied that there are fuzzy systems which can form one-to-one mappings from a multi-dimensional discrete space to a one-dimensional discrete space. Based on this and following the same idea as the approximation capability analysis of SNNs, the following theorem related to the approximation capability of SHNFSs can be proved as given in the Appendix.

Theorem 4 (Universal Approximation Property of SHNFSs). Let $G(X)$ be a function on discrete space $U = U_1 \times U_2 \times ... \times U_n$ in which $U_i = \{u_{i,k} \mid u_{i,k} \in R, k = 1,2,..., N_i\}$ $(i = 1,2,..., n)$. Then for any given $\varepsilon > 0$, there exists a SHNFS $SHNFS(X)$ given in (16) such that

$$\| G - SHNFS \|_\infty = \max_{X \in U} | G(X) - SHNFS(X) | < \varepsilon \quad (22)$$

Similarly, based on the fact mentioned above that there are one-to-one fuzzy systems on a multi-dimensional discrete space and following the same idea as the approximation capability analysis of ESNNs, the following theorem of the approximation capability of HNFSs can be proved as given in the Appendix.

Theorem 5 (Universal Approximation Property of HNFSs). Let $G(X)$ be a function on discrete space $U = U_1 \times U_2 \times ... \times U_n$ in which $U_i = \{u_{i,k} \mid u_{i,k} \in R, k = 1,2,..., N_i\}$ $(i = 1,2,..., n)$. Then for any given $\varepsilon > 0$ and for any disjoint grouping of the input variables $\{x_1, x_2,..., x_n\}$ into m groups G_j $(j = 1,2,..., m)$ satisfying (8)–(10), there exists a HNFS $HNFS(X)$ given in (19) such that

$$\| G - HNFS \|_\infty = \max_{X \in U} | G(X) - HNFS(X) | < \varepsilon \quad (24)$$

The two algorithms proposed for SNNs and ESNNs can be extended to identify SHNFSs and HNFSs. The main ideas are the same but there are several differences.

In the first algorithm, the lower level one-to-one linear mapping(s) are now replaced by the fuzzy system(s) as the lower level sub-system(s). As there are more parameters available to construct the one-to-one mapping(s), then it is possible that nonlinear one-to-one mappings can be constructed to allow the higher level approximation problem to become simpler. In addition, human knowledge can be utilized during the construction of the lower level fuzzy system(s).

In the second algorithm, rather than use the linear function(s) with parameters to be identified by the gradient descent optimisation algorithms, it is possible to construct fuzzy systems by using available human knowledge which may lead to faster convergence during the training phase.

In addition to the above, a third possible algorithm which is especially suitable for situations with high dimension and limited available data is as follows:

1. Construct one or several lower one-to-one fuzzy systems based on human knowledge to aggregate the impact of different input variable groups on the system output into several aggregated group indexes [i.e., construct $z_j = F_j(X_j)$ $(j = 1,2,...,m)$ by only using available human knowledge, and z_j is the aggregated index variable of those input variables in group G_j].
2. Use the constructed lower level fuzzy system(s) to transform the available input-output data $\{(X_t, y_t) \mid t = 1,2,...,N\}$ into the index-output data as $\{(Z_t, y_t) \mid Z_t = (z_{1,t},..., z_{m,t}),\ z_{j,t} = F_j(X_{j,t}),\ j = 1,2,..., m,\ t = 1,2,...,N\}$.
3. Use the index-output data $\{(Z_t, y_t) \mid t = 1,2,...,N)$ to identify the higher level NN $y = NN_m(Z)$ by using the NN learning algorithms.

A simple example is given to illustrate the meaning of the above steps. Suppose we wish to model how student performance in examinations is dependent on 9 study factors as follows: time spent in study, lecture attendance, homework completion, previous examination record, A-level scores, IQ score, lecture quality, lab facilities, and lab availability, based on collected data of a small number of students, say 25 (collecting such private information from a large group is costly and time consuming, and thus impractical). Suppose that the 9 factors are divided into 3 groups where Group 1 is the *effort* factors (time spent in study, lecture attendance, homework completion), Group 2 is the *academic ability* factor (previous examination record, A-level scores, IQ score), and Group 3 is the s*tudy environment* factors (lecture quality, lab facilities, and lab availability). Then the above three steps can be applied as follows:

Firstly, use human knowledge to build the lower level fuzzy subsystems. For example, the sub-system to aggregate the effort factors can be formed based on the following human knowledge fuzzy rule: *if time spent is long, lecture attendance is regular, homework completion is good, then the effort is very good*; such rules can form the effort index fuzzy sub-system.

Secondly, use the above lower level fuzzy systems to transform the input-output data into index-output data. For example, an input-output pair in the available data is {*[time spent=long, lecture attendance=regular, homework completion=good, …, lab availability=always], exam performance=good*}. Then use the lower level fuzzy sub-systems to transform the input-output pair into the index-output pair as {*[effort=very good, ..., study condition=good], exam performance=good*}. This step is to transform all input-output data to index-output data;

Thirdly, now the original modeling problem with 9 input variables and 25 available data has been transformed to a modeling problem with 3 input variables and 25 available data which is much easier to identify, thus a reasonable model is likely to be obtained as the limited training data is reasonably rich for a 3-dimensional modeling problem.

In summary, by using human knowledge to form the lower level fuzzy sub-systems and then transforming the available input-output data into index-output data, the modeling problem of a SHNFS or HNFS with n input variables is transformed into a modeling problem of a standard NN with m input variables. As the latter problem is one with a lower or much lower dimension, then it can be identified by using the existing learning algorithms based on the limited available data. In other words, the above proposed algorithm shows that SHNFSs and HNFSs have the potential to combine human (knowledge and experience) and machine intelligence (learning from data) to model high dimensional complicated systems with limited input-output data.

IV. Conclusion

This paper has investigated function approximation on discrete input spaces using neural networks and neural-fuzzy systems.

Firstly, from a hierarchical systematic view, this paper has proposed four new and simpler neural networks and hierarchical neural-fuzzy systems for function approximation on discrete spaces: SNNs, ESSNs, SHNFSs and HNFSs. Compared to standard NNs and fuzzy systems, the proposed approximation schemes have several advantages including being simpler (fewer parameters), useful to overcome model overfitting and underfitting, flexible, capable of utilizing both human (knowledge and experience) and machine intelligence (learning from data) for a difficult modeling situation (such as high dimensions and limited training data).

Secondly, the paper has analyzed the approximation capabilities of the proposed new approximation schemes. That is, whether the proposed approximation schemes preserve the universal approximation property of standard NNs and fuzzy systems. A positive answer to this question has been obtained, that is, all four proposed approximation schemes have the universal approximation property. These results have established a theoretical foundation and show the feasibility and general applicability of the proposed approximation schemes to function approximation on discrete spaces.

Thirdly, several possible algorithms have been proposed and analyzed to show how the advantages of the proposed approximation schemes can be realized.

Further work includes implementation and experimentation of the proposed algorithms, a comparison of the results obtained by the proposed algorithms with those obtained by standard NN learning algorithms, and applying these algorithms in real life applications.

Appendix

Proof of Theorem 1[35]. Given $U = U_1 \times U_2 \times ... \times U_n$ and $U_i = \{u_{i,k} \mid u_{i,k} \in R,\ k = 1,2,...,N_i\}$ $(i = 1,2,...,n)$, Without loss of generality, it is assumed that $u_{i,1} < u_{i,2} < ... < u_{i,N_i}$ $(i = 1,2,...,n)$. Now define

$$\Delta_i = \min_{1 \le k \le N_1 - 1}\{u_{i,k+1} - u_{i,k}\}$$

$$u_{i,N_i+1} = u_{i,N_i} + \Delta_i \qquad i = 1,...,n$$

and a linear function in the following:

$$y = M(X) = w_1\left(\frac{x_1 - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) + w_2\left(\frac{x_2 - u_{2,1}}{u_{2,N_2+1} - u_{2,1}}\right) + ... + w_n\left(\frac{x_n - u_{n,1}}{u_{n,N_n+1} - u_{n,1}}\right) \qquad (A.1)$$

in which the weighting factors w_i $(i = 1,2,...,n)$ are constructed recursively as follows:

$$w_1 = 1 \qquad 0 < w_{i+1} < w_i \frac{\Delta_i}{u_{i,N_i+1} - u_{i,1}} \qquad i = 1,...,n-1$$

For the above construction of the weighting factors w_i $(i = 1,2,...,n)$, it is implied that, for any given i, k and l $(i = 1,2,...,n-1, k = 1,2,...,N_i$ $l = 1,2,...,N_{i+1})$

$$\begin{aligned} & w_i \frac{u_{i,k} - u_{i,1}}{u_{i,N_i+1} - u_{i,1}} + w_{i+1} \\ & < w_i \frac{u_{i,k} - u_{i,1}}{u_{i,N_i+1} - u_{i,1}} + w_i \frac{u_{i,k+1} - u_{i,k}}{u_{i,N_i+1} - u_{i,1}} \qquad (A.2) \\ & = w_i \frac{u_{i,k+1} - u_{i,1}}{u_{i,N_i+1} - u_{i,1}} \le w_i \end{aligned}$$

Let and X_0 be $X_0^{'}$ any two different elements in U, that is, $X_0 = \left(u_{1,k_1}, u_{2,k_2}, ..., u_{n,k_n}\right)$ $X_0^{'} = \left(u_{1,k_1^{'}}, u_{2,k_2^{'}}, ..., u_{n,k_n^{'}}\right)$, and $X_0 \ne X_0^{'}$.If $u_{1,k_1} \ne u_{1,k_1^{'}}$, then, without loss of generality, assume that $u_{1,k_1} < u_{1,k_1^{'}}$ (this means $u_{1,k_1+1} \le u_{1,k_1^{'}}$). Now from (A.1) and (A.2) , it is implied that

$$\begin{aligned} M(X_0) &= w_1\left(\frac{u_{1,k_1} - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) + w_2\left(\frac{u_{2,k_2} - u_{2,1}}{u_{2,N_2+1} - u_{2,1}}\right) \\ &\quad + ... + w_n\left(\frac{u_{n,k_n} - u_{n,1}}{u_{n,N_n+1} - u_{n,1}}\right) \\ &< w_1\left(\frac{u_{1,k_1} - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) + w_2\left(\frac{u_{2,k_2} - u_{2,1}}{u_{2,N_2+1} - u_{2,1}}\right) \\ &\quad + ... + w_{n-1}\left(\frac{u_{n-1,k_{n-1}} - u_{n-1,1}}{u_{n-1,N_{n-1}+1} - u_{n-1,1}}\right) + w_n \\ &< w_1\left(\frac{u_{1,k_1} - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) + w_2\left(\frac{u_{2,k_2} - u_{2,1}}{u_{2,N_2+1} - u_{2,1}}\right) \\ &\quad + ... + w_{n-2}\left(\frac{u_{n-2,k_{n-2}} - u_{n-2,1}}{u_{n-2,N_{n-2}+1} - u_{n-2,1}}\right) + w_{n-1} \qquad by\ (A.2) \\ &< w_1\left(\frac{u_{1,k_1} - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) + w_2 \qquad by\ (A.2) \\ &< w_1\left(\frac{u_{1,k_1+1} - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) \qquad by\ (A.2) \\ &\le w_1\left(\frac{u_{1,k_1^{'}} - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) \\ &\le w_1\left(\frac{u_{1,k_1^{'}} - u_{1,1}}{u_{1,N_1+1} - u_{1,1}}\right) + w_2\left(\frac{u_{2,k_2^{'}} - u_{2,1}}{u_{2,N_2+1} - u_{2,1}}\right) \\ &\quad + ... + w_n\left(\frac{u_{n,k_n^{'}} - u_{n,1}}{u_{n,N_n+1} - u_{n,1}}\right) = M(X_0^{'}) \end{aligned}$$

That is, $M(X_0) \ne M(X_0^{'})$. If $u_{1,k_1} = u_{1,k_1^{'}}$ but i_0 is the smallest i such that $u_{i_0,k_{i_0}} \ne u_{i_0,k_{i_0}^{'}}$, then similar to the above it can be proved that $M(X_0) \ne M(X_0^{'})$. Therefore, if $X_0 \ne X_0^{'}$, then $M(X_0) \ne M(X_0^{'})$. That is, the linear function given in (A.1) is a one-to-one mapping from U to R.

Proof of Theorem 2. For the given input space U, based on Theorem 1, there exists a linear function

$$z = L(X) = w_0 + \sum_{i=1}^{n} w_i x_i \qquad (A.3)$$

which is a one-to-one mapping from U to R. For every

$$X_{k_1 k_2 ... k_n} = \left(u_{1,k_1}, u_{2,k_2}, ..., u_{n,k_n}\right) \in U = \mathop{\times}_{i=1}^{n} U_i$$

$$k_i = 1,2,...,N_i \qquad l = 1,2,...,n$$

define

$$z_{k_1k_2\ldots k_n} = L\left(X_{k_1k_2\ldots k_n}\right)$$

That is, $z_{k_1k_2\ldots k_n}$ is the function value of $L(X)$ at $X_{k_1k_2\ldots k_n}$ and the set of all such values is denoted as

$$V = \left\{y_{k_1k_2\ldots k_n} \,\middle|\, k_l = 1,2,\ldots,N_i, i = 1,2,\ldots,n\right\}$$

which is the output variable space of function $L(X)$. As $L(X)$ is a one-to-one mapping, then all elements of V are different. Therefore, for every $z \in V$, there exists only one element X in U such that $z = L(X)$. Further, as U is a discrete space with finite elements, then V is a discrete space with finite elements.

Now define function $g(z)$ on V as follows: For every $z \in U$, let X be the unique element in U such that $z = L(X)$. Then define the value of g at z as follows:

$$g(z) = G(X)$$

For the function g defined in the above, it can be proved by the reverse process that for all $X \in U$

$$G(X) = g[L(X)] \tag{A.4}$$

As $g(z)$ is a function on finite discrete space V which is bounded, based on [27] it can extended to be a continuous function $\hat{g}(X)$ on $\hat{V} = [\underline{z}, \overline{z}]$ (where $\underline{z} = \min_{z \in V} z$, $\overline{z} = \max_{z \in V} z$) in the sense that

$$\hat{g}(X) = g(X) \qquad z \in V \tag{A.5}$$

As $\hat{g}(X)$ is a continuous function on $\hat{V}$, then it is implied immediately from the universal approximation property of standard NNs on continuous spaces that there exists a NN $NN_1(z)$ on $\hat{U}$ such that

$$\| \hat{g} - NN_1 \|_\infty = \max_{z \in \hat{V}} | \hat{g}(z) - NN_1(z) | < \varepsilon \tag{A.6}$$

Now define a SNN as $SNN(X) = NN_1[L(X)]$, then (A.4), (A.5) and (A.6) imply that, for any $X \in U$,

$$\begin{aligned} &| G(X) - SNN(X) | = | g[L(X)] - NN_1[L(X)] | \\ &\le \max_{z \in V} \left| g(z) - NN_1(z) \right| \\ &\le \max_{z \in \hat{V}} | \hat{g}(z) - NN_1(z) | < \varepsilon \end{aligned}$$

which leads to (20) immediately and this completes the proof.

Proof of Theorem 3. For each $j = 1,2,\ldots,m$, based on Theorem 1, there exists a linear function defined on $U_{G_j} = \times_{k=1}^{n_j} U_{i_k}$ as follows

$$z_j = L_j(X_j) = w_{j,0} + \sum_{k=1}^{n_j} w_{j,k} x_{i_k^{(j)}} \tag{A.7}$$

which is a one-to-one mapping from U_{G_j} to R. For every

$$X_j^{(k_1k_2\ldots k_{n_j})} = \left(u_{i_1,k_1}, u_{i_2,k_2}, \ldots, u_{i_{n_j},k_{n_j}}\right) \in U_{G_j}$$
$$k_l = 1,2,\ldots,N_{i_l} \quad l = 1,2,\ldots,n_j$$

define

$$z_j^{(k_1k_2\ldots k_{n_j})} = L_j\left(X_j^{(k_1k_2\ldots k_{n_j})}\right)$$

That is, $z_j^{(k_1k_2\ldots k_{n_j})}$ is the function value of $L_j(X_j)$ at $X_j^{(k_1k_2\ldots k_{n_j})}$ and the set of all such values is denoted as

$$V_j = \left\{z_j^{(k_1k_2\ldots k_{n_j})} \,\middle|\, k_l = 1,2,\ldots,N_{i_l}, l = 1,2,\ldots,n_j\right\}$$

which is the output variable space of function $L_j(X_j)$. As $L_j(X_j)$ is one-to-one mapping, then all elements of V_j are different. Therefore, for every $z_j \in V_j$, there exists only one element X_j in U_{G_j} such that $z_j = L_j(X_j)$. Further, as U_{G_j} is a discrete space with finite elements, then V_j is a discrete space with finite elements.

Now define function $g(Z) = g(z_1, z_2, \ldots, z_m)$ on $V = \times_{j=1}^{m} V_j$ as follows: for any given $Z = (z_1, \ldots, z_m) \in V$, as each $z_j \in V_j$ $(j = 1,2,\ldots,m)$, then there exists a unique element X_j in U_{G_j} such that $z_j = L_j(X_j)$. Further it can be implied from (8)-(10) that all sub-vectors X_j $(j = 1,2,\ldots,m)$ form a unique vector $X = (x_1, \ldots, x_n) \in U$. Now define the value of g at the given $Z = (z_1, \ldots, z_m) \in V$ as the value of G at its unique corresponding point $X = (x_1, \ldots, x_n) \in U$. That is

$$g(Z) = g(z_1, \ldots, z_m) = G(X)$$

For the function $g(Z)$ defined in the above, it can be proved by the reverse process that for all $X = (x_1, \ldots, x_n) \in U$,

$$G(X) = g[L_1(X_1), \ldots, L_m(X_m)] \tag{A.8}$$

As $g(Z)$ is a function on a finite discrete space V which is bounded, then, from the fact that any function in a discrete space can be extended to be a continuous function [27], it is implied that $g(Z)$ can be extended to a continuous function $\hat{g}(Z)$ on $\hat{V} = \times_{j=1}^{m} [\underline{z}_j, \overline{z}_j]$ [where $\underline{z}_j = \min_{z_j \in V_j} z_j$, $\overline{z}_j = \max_{z_j \in V_j} z_j$ $(j = 1,2,\ldots,m)$] in the sense that $\hat{g}(Z) = g(Z)$ for any $Z \in V = \times_{j=1}^{m} V_j$. As $\hat{g}(Z)$ is a continuous function on $\hat{V}$, then it is implied immediately from the universal approximation property of standard NNs on continuous spaces that there exists a NN $NN_m(Z) = NN_m(z_1,...,z_m)$ on $\hat{V}$ such that

$$\| \hat{g} - NN_m \|_\infty = \max_{Z\in\hat{V}} | \hat{g}(Z) - NN_m(Z) | < \varepsilon \quad \text{(A.9)}$$

Now define the ESNN as

$$ESNN(X) = NN_m[L_1(X_1),...,L_m(X_m)]$$

This, together with (A.8) and (A.9), implies that, for any $X \in U$,

$$\begin{aligned}
&| G(X) - ESNN(X) | \\
&= | g[L_1(X_1),...,L_m(X_m)] - NN_m[L_1(X_1),...,L_m(X_m)] | \\
&\le \max_{(z_1,...,z_m)\in V} \left| g(z_1,...,z_m) - NN_m(z_1,...,z_m) \right| \\
&= \max_{Z\in V} \left| g(Z) - NN_m(Z) \right| \\
&\le \max_{z\in\hat{V}} | \hat{g}(Z) - NN_m(Z) | < \varepsilon
\end{aligned}$$

which leads to (21) immediately and this completes the proof.

Proof of Theorem 4. For the given input space U, based on Theorem 1, there exists a linear function

$$z = L(X) = w_0 + \sum_{i=1}^{n} w_i x_i \quad \text{(A.3)}$$

which is a one-to-one mapping from U to R. For the given $L(X)$, based on Theorem 4 in [32], it can be implied that there exists a simplest fuzzy system $F(X)$ [i.e., there are only two memberships in each $U_i (i = 1,2,...,n)$] such that $F(X) = L(X)$ for all $X \in U$. Then it is implied that, from the fact that $L(X)$ is one-to-one mapping, the fuzzy system $F(X)$ is a one-to-one mapping from U to R.

Based on this one-to-one fuzzy mapping $F(X)$, the rest of the proof is the same as the proof of Theorem 2 except for replacing $z = F(X)$ by $z = L(X)$ and therefore the details are omitted.

Proof of Theorem 5. From the proof of Theorem 4, it is obtained that, for any given discrete space U as in (2), there exists a one-to-one fuzzy system from U to R. Applying this result to the input space of each group, i.e., $U_{G_j} = \times_{k=1}^{n_j} U_{i_k}$ $(j = 1,2,...,m)$, we can obtain that, for each U_{G_j}, there exists a fuzzy sub-systems $z_j = F_j(X_j)$ which is a one-to-one mapping from U_{G_j} to R. Based on this, the proof of the theorem is the same as the proof of Theorem 3 except for replacing $z_j = L_j(X_j)$ by $z_j = F_j(X_j)$ $(j = 1,2,...,m)$ and so the details are omitted.

References

[1] A. R. Barron, “Universal approximation bounds for superpositions of a sigmoidal function,” *IEEE Trans. Inform. Theory*, vol. 39, pp. 930–945, 1993.

[2] J.J. Buckley, “Universal fuzzy controllers,” *Automatica*, Vol. 28, pp. 1245-1248, 1992.

[3] S. Carroll, and B. Dickinson, “Construction of neural networks using the Radon transform,” in *IEEE International Conference on Neural Networks*, vol. 1. Washington, DC: IEEE, pp. 607–611, 1989.

[4] J.L. Castro, “Fuzzy logic controllers are universal approximators,” *IEEE Trans. Syst., Man, Cybern.*, Vol. 25, pp. 629-635, 1995.

[5] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals, and Systems*, Vol. *3*, pp. 303–314, 1989.

[6] B. Delyon, A. Juditsky, and A. Benveniste, “Accuracy analysis for wavelet approximations,” *IEEE Trans. Neural Networks*, Vol. 6, pp. 332-348, 1995.

[7] G. Feng, “Controller Synthesis of Fuzzy Dynamic Systems Based on Piecewise Lyapunov Functions”, *IEEE Trans. Fuzzy Syst.*, Vol. 11, pp. 605–612, 2003.

[8] S. Ferrari, M. Maggioni, and N. A. Borghese, “Multiscale approximation with hierarchical radial basis functions networks,” *IEEE Trans. Neural Networks*, Vol. 15, pp. 178-188, 2004

[9] B. Hammer and Kai Gersmann, “A Note on the universal approximation capability of support vector machines,” *Neural Processing Letters*, Vol. 17, pp. 43–53, 2003.

[10] R. Hassine, F. Karray, A. M. Alimi, and M. Selmi, “Approximation properties of fuzzy systems for smooth functions and their first-order derivative,” *IEEE Trans. Syst., Man, Cybern.-Part A*, Vol. 33, pp. 160-168, 2003.

[11] R. Hecht-Nielsen, “Kolmogorov’s mapping neural network existence theorem,” In *IEEE International Conference on Neural Networks*, vol. 3. Washington, DC: IEEE, pp. 11–14, 1989.

[12] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators, Neural Networks,” *Neural Networks*, Vol. 2, pp. 359–366, 1989.

[13] K. Hornik, “Some results on neural network approximation,” *Neural Networks*, Vol. 6, pp. 1069–1072, 1993.

[14] B. Igelnik and N. Parikh, “Kolmogorov’s Spline Network,”, *IEEE Trans. Neural Networks*, Vol. 14, pp. 725-733, 2003

[15] Y. Ito, “Approximation of functions on a compact set by finite sums of sigmoid function without scaling,” *Neural Networks*, Vol. 4, pp. 817–826.

[16] B. Kosko, “Fuzzy systems as universal approximators,” in *Proc. of IEEE int. conf. on Fuzzy Systems*, San Diego, CA, pp. 1153—1162, 1992..

[17] V. Kreinovich, “Arbitrary nonlinearity is sufficient to represent all functions by neural networks: a theorem,” *Neural Networks*, Vol. 4, pp. 381–383, 1991.

[18] V. Kurkova, “Kolmogorov’s theorem and multilayer neural Networks,” *Neural Networks*, Vol. 5, pp. 501–506, 1992.

[19] A. Mencattini, M. Salmeri, and A. Salsano, “Sufficient conditions to impose derivative constraints on MISO Takagi–Sugeno fuzzy logic systems,” *IEEE Trans. Fuzzy Syst.*, vol. 13, pp. 454–467, Aug. 2005.

[20] T. Poggio and F. Girosi, “Networks for approximation and learning,” *Proc. of IEEE*, Vol. 78, pp. 1481-1497, Sept. 1990.

[21] F. Scarselli and A. C. Tsoi, “Universal approximation using feedforward neural networks: A survey of some existing methods and some new results,” *Neural Networks*, vol. 11, pp. 15–37, 1998.

[22] V. Torra, “A Review of the construction of hierarchical fuzzy systems,” *Int. J. Intelligent Systems*, Vol. 17, pp. 531-543, 2002.

[23] L. Vecci, F. Piazza, and A Uncini, “Learning and approximation capabilities of adaptive spline activation function neural networks”, *Neural Networks*, vol. 11, pp. 259–270, 1998.

[24] L.-X. Wang, “Fuzzy systems are universal approximators,” in *Proc. of IEEE int. conf. on Fuzzy Systems*, San Diego, CA, pp. 1163-1170, 1992.

[25] L.-X. Wang, “Universal approximation by hierarchical fuzzy systems,” *Fuzzy Sets and Systems*, Vol. 93, pp. 223-230, 1998.

[26] L.-X. Wang, “Analysis and design of hierarchical fuzzy systems,” *IEEE Trans. Fuzzy Systems*, Vol.7, pp. 617-624, 1999.

[27] G. A. Watson, *Approximation Theory and Numerical Methods*, New York:John Wiley and Sons, 1980.

[28] H. Ying, “Sufficient conditions on general fuzzy systems as function approximations,” *Automatica*, Vol. 30, pp. 521-525, Mar. 1994.

[29] X.-J. Zeng and M.G. Singh, “Approximation theory of fuzzy systems---SISO case,” *IEEE Trans. Fuzzy Systems*, Vol. 2, pp. 162-176, May 1994.

[30] X.-J. Zeng and M.G. Singh, “Approximation theory of fuzzy systems---MIMO case,” *IEEE Trans. Fuzzy Systems*, Vol. 3, pp. 219-235, May 1995.

[31] X.-J. Zeng and M.G. Singh, “Approximation accuracy analysis of fuzzy systems as function approximators,” *IEEE Trans. Fuzzy Systems*, Vol. 4, pp. 44-63, Feb. 1996.

[32] X.-J. Zeng and M.G. Singh, “Decomposition property of fuzzy systems and its applications,” *IEEE Trans. Fuzzy Systems*, Vol. 4, pp. 149-165, Apr. 1996.

[33] X.-J. Zeng and J.A. Keane, “Approximation capabilities of hierarchical fuzzy systems,” *IEEE Trans. Fuzzy Systems*, to be published.

[34] X.-J. Zeng and J.A. Keane, “Approximation capabilities of hierarchical hybrid systems,” *IEEE Trans. Syst., Man, Cybern.-Part A*, to be published

[35] X.-J. Zeng, J.A. Keane, John Yannis Goulermas, and Panos Liatsis, “Hierarchical fuzzy systems for function approximation on discrete input spaces ,” submitted for publication.

Author Biographies

Xiao-Jun Zeng received the B.Sc. degree in mathematics and the M.Sc. degree in computer and systems sciences from Xiamen University, Xiamen, China. He received the Ph.D. degree in computation from the University of Manchester Institute of Science and Technology (UMIST), Manchester, U.K.

He is currently a lecturer in the School of Informatics, University of Manchester. From 1996 to 2002, he was with the Knowledge Support Systems Ltd., Manchester, U.K., where he was a scientific developer, senior scientific developer, and head of research. From 1985 to 1992, he was with the Department of Computer and Systems Sciences, Xiamen University, where he was a lecturer and an associate professor. His current research interests include fuzzy systems, neural networks, decision support systems, intelligent systems, and data mining.

Dr. Zeng is an Associate Editor of the *IEEE Transactions on Fuzzy Systems* and is a member of the editorial board of the *International Journal of Computational Intelligence Research*.

John Yannis Goulermas was born in Greece in 1970. He received the B.Sc. degree (Hons, Class I) in computation from the University of Manchester (UMIST), Manchester, U.K., in 1994. In 1996 and 2000, he received the M.Sc. degree by research and the Ph.D. degree from the Control Systems Centre, Department of Electrical Engineering and Electronics (EE&E) at UMIST working in the area of Machine Vision.

He has worked for two years in industry in the area of financial/pricing modelling and optimization, and for three years in the Centre for Virtual Environments and the Centre for Rehabilitation and Human Performance Research of the University of Salford, as a Senior Research Fellow in the area of biomechanics and intelligent gait analysis. He is currently a lecturer in the EE&E department at the University of Liverpool, U.K. His main research interests include pattern recognition, neural networks, data analysis, artificial intelligence, machine vision and optimization.

John A. Keane received the B.Sc. and MSc degrees in computation and computer science respectively.

He holds the M.G. Singh Chair in Computing Science in the School of Informatics at the University of Manchester, UK, where he leads the Data and Decision Engineering research group. He has worked in industry with the Trustees Savings Bank, Philips Data Systems and ICL. His research activity is in the areas of data intensive systems, data mining, and parallel systems.

Professor Keane is Deputy Director of the UK National Centre for Text Mining which is led by the Universities of Manchester, Liverpool and Salford, and involves internationally self-funded partners University College, Berkeley, University of Geneva, University of Tokyo, and the San Diego Supercomputer Centre.

Professor Keane is an Associate Editor of the *IEEE Transactions on Systems, Man and Cybernetics Part C,* a member of the editorial board of *Simulation Modelling,* and is a member of the EPSRC Peer Review College.

Panos Liatsis received the Dipl.Eng. degree (first-class hons.) in electrical engineering from the University of Thrace, Thrace, Greece and the Ph.D. degree from the Control Systems Centre, University of Manchester Institute of Science and Technology (UMIST), Manchester, U.K.

After working as a Lecturer in the Control Systems Centre, UMIST, he moved to the School of Engineering and Mathematical Sciences at City University, London, U.K., where he is currently a Senior Lecturer and Director of the Information and Biomedical Engineering Centre (IBEC). His main research interests are neural networks, genetic algorithms, computer vision and pattern recognition. He has worked in the areas of intelligent automotive sensors for object tracking, collision warning, lane support and traffic sign recognition, feature/area-based stereo matching using co-evolutionary computing, polynomial neural networks architectures for predictive image coding and time series forecasting, kernel-based discriminant analysis for chemometrics in biomedical signal processing, among many others. He has published over 80 scientific papers in international journals and conferences and edited two international conference proceedings.

Dr Liatsis is a member of the EPSRC Peer Review College, the IEE, the Technical Chamber of Greece (TEE), and a European Engineer (Eur Ing).