

## Enhanced hybrid method of divide-and-conquer and RBF neural networks for function approximation of complex problems

Mohammed AWAD\*

Department of Computer Systems Engineering, Faculty of Engineering and Information Technology,  
Arab American University, Jenin, Palestine

Received: 12.10.2015

Accepted/Published Online: 24.03.2016

Final Version: 10.04.2017

**Abstract:** This paper provides an enhanced method focused on reducing the computational complexity of function approximation problems by dividing the input data vectors into small groups, which avoids the curse of dimensionality. The computational complexity and memory requirements of the approximated problem are higher when the input data dimensionality increases. A divide-and-conquer algorithm is used to distribute the input data of the complex problem to a divided radial basis function neural network (Div-RBFNN). Under this algorithm, the input data variables are typically distributed to different RBFNNs based on whether the number of the input data dimensions is odd or even. In this paper, each Div-RBFNN will be executed independently and the resulting outputs of all Div-RBFNNs will be grouped using a system of linear combination function. The parameters of each Div-RBFNN (centers, radii, and weights) will be optimized using an efficient learning algorithm, which depends on an enhanced clustering algorithm for function approximation, which clusters the centers of the RBFs. Compared to traditional RBFNNs, the proposed methodology reduces the number of executing parameters. It further outperforms traditional RBFNNs not only with respect to the execution time but also in terms of the number of executing parameters of the system, which produces better approximation error.

**Key words:** Approximation algorithms, computational complexity, divide-and-conquer algorithm, RBF neural networks

### 1. Introduction

Developing multivariate models for industrial or medical applications produces a computational complexity problem [1,2]. The complexity of the input data affects the quality of the neural network training process. Often, such complexity compromises the accuracy of the approximation process [3,4]. Other data complexity factors such as noise, atypical patterns, overlap, and bad data distribution usually weaken the quality of the training process, too [5]. The curse of dimensionality problem increases the computational complexity and memory requirements, in some cases exponentially [1,2]. Due to the increased number of input data variables, the number of executing parameters usually increases exponentially. One of the instrumental methods used to tackle the problem is the divide-and-conquer algorithm, which is commonly used to attenuate the computational complexity of the input data variables [6,7]. A divide-and-conquer algorithm, used to reduce the dimension of the input data variables by dividing the input data variables and allowing the learning algorithms to be executed effectively, breaks the input data into simpler sets before grouping the subsets and providing a solution. This philosophy of dividing the input data governs the operation of the neural network model. The topology of

\*Correspondence: mohammed.awad@aauj.edu

neural networks is produced by the split process, with different modules assigning different data to a variety of regions. The learning process, using divide-and-conquer algorithms, combines the output of each divided neural network, thus allowing data to divide the space into subregions and, where possible, handling the training of complex data using neural networks.

Owing to its simple topology and fast learning algorithms, the radial basis function neural network (RBFNN) [8,9] is one of the most common types of neural networks. The RBFNN usually uses a Gaussian activation function in the hidden layer; the basic topology of a RBFNN is a 3-layer neural network, as illustrated in Figure 1. The RBF  $\Phi$  can be calculated using a Gaussian function as in the following expression:

$$\Phi(\vec{x}, \vec{c}, r) = \exp\left(-\frac{\|\vec{x} - \vec{c}\|^2}{r}\right), \tag{1}$$

where  $\vec{x}$  is the input data variables,  $\vec{c}$  is the central point of the function  $\Phi$ , and  $r$  is its radius.

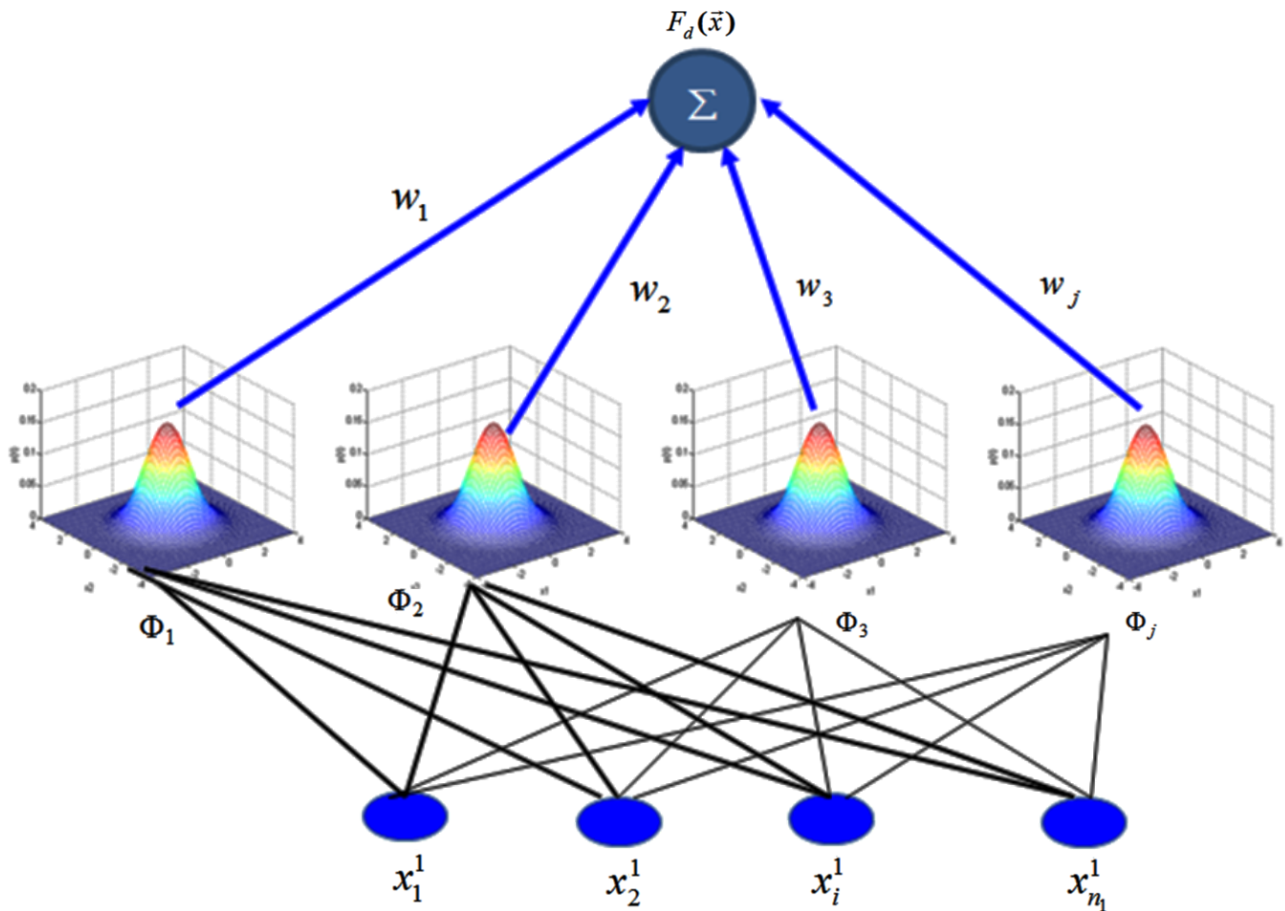


Figure 1. Radial basis function neural network.

The output of the RBFNN is given by the following expression:

$$F_d(\vec{x}, \Phi, w) = \sum_{i=1}^m \Phi_i(\vec{x}) \cdot w_i, \tag{2}$$

where  $\Phi = \{\Phi_i : i = 1, \dots, m\}$  are the basis functions set and  $w_i$  denotes the weights of the RBF.

In general, the complexity of RBFNN topology depends on the number of neurons (RBF) used in the hidden layer, which, in turn, depends on the number of input data variables. A decrease in input data variables can reduce the number of execution parameters, which produces a fast and effective learning process. One of the most frequently used methods to reduce the number of execution parameters is principal component analysis (PCA). Another method is independent component analysis (ICA), which is typically used with neural networks and signal processing [10,11]. Several related works have also utilized PCA and ICA to decrease the number of execution parameters in neural networks meant to solve function approximation problems. Pomares et al. [12] obtained the structure of a complete rule-based fuzzy system for specific approximation accuracy of the training data, deciding which input variables should be taken into account and the number of membership functions needed in every selected input variable in order to reach the approximation target. Vehtari and Lampinen [1] proposed the use of posterior and marginal posterior probabilities obtained via variable-dimension Markov chain Monte Carlo methods to find out potentially useful input combinations and to perform the final model choice and assessment using the expected utilities computed by using the cross-validation predictive densities. For a given set of input and output variables, Chen and Wang [13] proposed using fuzzy partition, which maps fuzzy sets into each input variable. Cheung and Huang [14] used a divide-and-conquer-based learning approach to RBFNNs. They divided the conventional RBFNN into several RBF subnetworks, each of which individually takes a subspace of input data of each dimension as input. The original RBFNN output becomes a linear combination of the subnetworks' outputs, with the coefficients adaptively learned together with the system parameters of each subnetwork, a traditional learning algorithm used for optimizing centers, radii, and weight of each sub-RBFNN. The divide-and-conquer algorithm is a simple and successful method in solving problems with large input data dimensions. Its idea is derived from breaking large amounts of input data into smaller sets, solving the divided subproblems separately, and then combining the subsolutions into a full solution [15]. Our goal is to reduce the computational complexity of function approximation problems using a RBFNN and divide-and-conquer algorithm. The proposed Div-RBFNN method generates an output vector for each input vector. The resulting products are simultaneously combined using a linear combination function, which also uses these values to optimize the centers, radii, and weights in the input space of each RBFNN so that the final output of the Div-RBFNN will be a weighted sum of all vector output. Local models operate separately but provide information to the output that can be strongly correlated, so that the overall system performance can be improved in terms of reliability and fault tolerance.

Overall Div-RBFNN system output is derived from a combination of the outputs, where the approximation of the total system, which is generally higher than any of the individual approximation Div-RBFNN topologies, provides a suitable construction of the RBFNN, which eventually improves the performance of complex function approximation problems.

In this paper, the proposed hybrid Div-RBFNN method is capable of modeling complex systems by reducing the number of execution parameters. The methodology divides the input data variables into equal parts if they are even and into equal parts plus one if they are odd. For each divided part, the traditional algorithms of enhanced clustering for function approximation (ECFA) [16] are used to optimize the parameters of each sub-Div-RBFNN; this algorithm depends on the calculation of the error committed in each cluster using the current output of the RBFNN and trying to concentrate more clusters in input regions where the approximation error is bigger, attempting to homogenize the contribution to the error of every cluster [16]. The number of RBFNNs depends on the number of the input data variables and on whether the value of input data dimensions is odd or even.

The paper is organized as follows: Section 2 describes the basic hybrid proposed model and topology

of the Div-RBFNN, as well as the traditional clustering method to optimize Div-RBFNN parameters, while Section 3 presents hypothetical numerical examples on how the proposed hybrid Div-RBFNN is capable of reducing execution parameters with the best approximation error.

## 2. Topology of the Div-RBFNN

In a traditional RBFNN, all neurons in the hidden layer receive input from all variables. The development in this proposed hybrid method focuses on studying the effect of creating a different topology of Div-RBFNN to solve the problem of computational complexity in neural network training. Div-RBFNN topology is particularly instrumental when the input data dimensions of the problem are divided into subproblems connected in parallel. Every sub-Div-RBFNN is a RBFNN. The entire sub-Div-RBFNN has the hybrid Div-RBFNN topology as a total output. The number of RBFNNs in the hybrid Div-RBFNN system depends on whether the values of input data dimensions are odd or even. The training parameters of each sub-RBFNN in the Div-RBFNN (centers, radii, and weight, and the number of neurons in each sub-Div-RBFNN) are optimized using an efficient algorithm of clustering designed for function approximation.

In order to observe the role of different topologies in solving function approximation problems, different models of Div-RBFNNs are used as a tool to generate RBFNN topologies depending on the input data dimensions, yet with a different structure. The success rates achieved by the Div-RBFNN structure were higher than that of an independent RBFNN employed with the Levenberg–Marquardt algorithm [17]. Div-RBFNN topology divides the problem into simpler problems and combines each solution. The Div-RBFNN training algorithm is suitable for modular structures and it converges faster than traditional RBFNNs. This hybrid method has been built differently to target the function and the neighborhood of solutions, allowing accurate observation of the influence of the Div-RBFNN topology on the generated solution fields. The contribution focuses on establishing a framework for the effective and practical model to study the effect of the Div-RBFNN topology of an instance of computational complexity problems. To arrive at this, the study proposes a measure that operates regardless of the size of the instance and the parameters of the problem. This measure can be applied in the study of the structural effects of instances of other computational problems and even by using other types of algorithms. We propose a hybrid method that reduces the input data variables and creates the most suitable topology of Div-RBFNN, as shown in Figure 2.

Each subset of the input variables  $\{x_1, \dots, x_n\}$  can be used as the input to each divided part (sub-Div-RBFNN). Each sub-Div-RBFNN receives its specific input variable values depending on the dimensions of the input data, as shown in the following recursive solution formula:

$$\# \text{ of input}_{dim} \text{ each RBFNN} = \begin{cases} dim/2 & \text{if } d = \text{even}. \\ (dim - 1)/2 & \text{if } d = \text{odd} \ \& \ d \geq 3. \end{cases} \quad (3)$$

Each sub-Div-RBFNN receives variable(s) and applies the process of optimization of its parameters (centers  $\vec{c}^d$ , radii  $r^d$ ). The optimization of the weight between hidden layer and output layer in the Div-RBFNN does not depend on the output of every sub-Div-RBFNN  $\{F_1(x), \dots, F_T(x)\}$ ; rather, it relies on the total output  $F_T(x)$  of the hybrid Div-RBFNN, as in the linear formula below.

$$F_T(\vec{x}, \Phi, w) = \sum_{d=1}^d \sum_{i=1}^{m_d} \Phi_i^d(\vec{x}) \cdot w_i^d \quad (4)$$

$w_i^d$  Here, are the weights and represents the  $i$ th basis functions of the  $d$ th RBFNN.

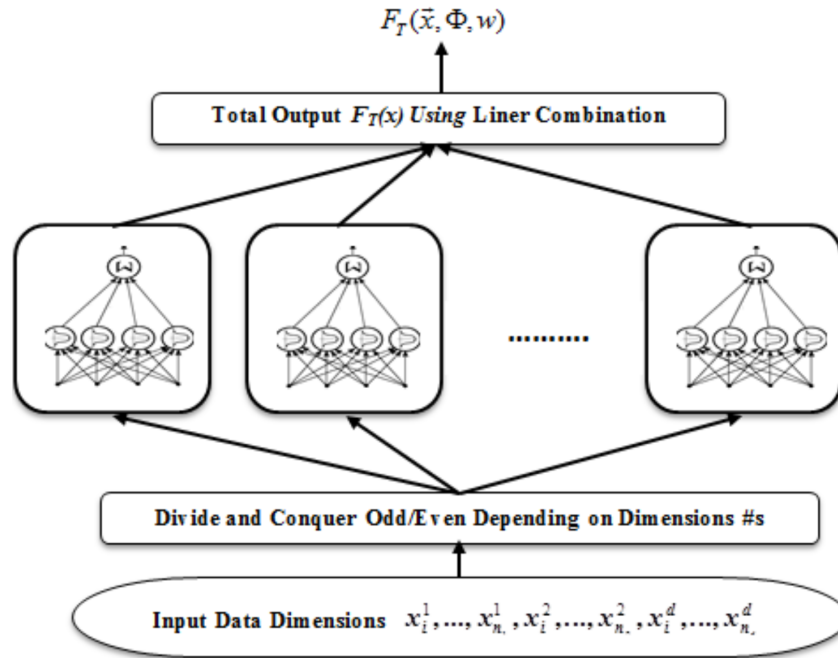


Figure 2. Div-RBFNN.

The divide-and-conquer processes are usually applied recursively. The efficiency of this technique depends on how the subdivisions are resolved. Generally speaking, however, the process depends on three stages:

**Division step:** The original problem is split into smaller subproblems,  $d$ .

**Conquer step:** The subproblems are solved independently, directly if they are simple, or reduced to more simple cases (typically recursively).

**Merge step:** The individual solutions are combined to obtain the solution of the original problem.

The most frequent situation is when  $d = 2$ .

Function divide and conquer

(input  $x_i^1, \dots, x_{n_1}^1, x_i^2, \dots, x_{n_2}^2, x_i^d, \dots, x_{n_d}^d$ ).

**Return** output  $F_{d1}(\vec{x}, \Phi, w), \dots, F_{dn}(\vec{x}, \Phi, w)$ );

**If** the input is  $\leq 1$ , then it returns a simple solution.

**Else** decompose input in  $x_i^1, \dots, x_{n_1}^1, x_i^2, \dots, x_{n_2}^2, x_i^d, \dots, x_{n_d}^d$

**For**  $i = 1$  to do

Y = divide and conquer ( $x_i$ ).

**End**

**Return** combined subcases solution.

The Div-RBFNN can be obtained for any given problem from a set of input variables. For example, for 4-input  $\{x_1, \dots, x_4\}$  even number of dimensions and 5-input  $\{x_1, \dots, x_5\}$  odd number of dimensions, two possible different topologies can be obtained for each case (even, odd). The Div-RBFNN configuration affects the number of executing parameters of the proposed hybrid system; the total number of parameters in every sub-Div-RBFNN can be illustrated by the following expression [8].

$$\# \text{ of parameters}_{Rbfnn} = m_d \cdot (n_d + 2) \tag{5}$$

Here,  $m_d$  is the number of RBFs in the  $d$ th sub-Div-RBFNN and  $n_d$  is the number of input variables actually used by this  $d$ th sub-Div-RBFNN. Table 1 shows the number of parameters used in each of these topologies using a total number of 20 RBFs for each one in the odd case and 25 RBFs for each one in the even case. It is clear that any topology of odd/oven Div-RBFNN produces an execution number of parameters lower than that of the traditional RBFNN with the same number of neurons in the hidden layer.

**Table 1.** Number of parameters used in each Div-RBFNN.

# of Inputs	# Sub-Div-RBFNNs	#RBF Sub-Div-RBFNN	#Var. Sub-Div-RBFNN	#Parm. Sub-Div-RBFNN	#Parm. Div-RBFNN
Even	4	5	1	15	60
		5	1	15	
		5	1	15	
		5	1	15	
	2	10	2	40	80
		10	2	40	
Odd	5	5	1	15	75
		5	1	15	
		5	1	15	
		5	1	15	
		5	1	15	
	2	10	2	40	115
		15	3	75	
	2	15	3	75	115
		10	2	40	
	3	10	2	40	95
		10	2	40	
		5	1	15	

The training process depends on an efficiently supervised algorithm of clustering designed for function approximation to initialize the parameters in each sub-Div-RBFNN. Figure 3 presents, in a schematic way, the general description of the proposed hybrid Div-RBFNN method. The learning process is informed by the minimization of an error function calculated as shown below.

$$Er_n^t = \frac{1}{2} \sum_{j=1}^t \sum_{i=1}^n (f(x_i^t, \Phi, w) - Y_i)^2 \tag{6}$$

Here,  $f((x_i^t, \Phi, w)$  is the total output  $F_T(x)$ , and  $Y_i$  is the real output. The number of RBFs increases in each iteration only by one RBF in one hybrid Div-RBFNN until there is no improvement in the test error during several iterations.

In the hybrid Div-RBFNN, a supervised algorithm of clustering is used for initializing the learning parameters in each sub-Div-RBFNN. This algorithm uses the target output for each input vector of the training data and calculates the error produced by each cluster in the real output of the approximated problem. This process results in an increasing number of clusters that migrate into zones where the cluster produces a bigger error by moving clusters that have a small error to zones of clusters that have a bigger error [16,18]. The next step of this clustering algorithm is fixing the value of the radius  $r^d$  of each cluster (RBF) through a heuristic algorithm of k-nearest neighbors [16]. Singular value decomposition [19,20] is used to find the optimal weight.

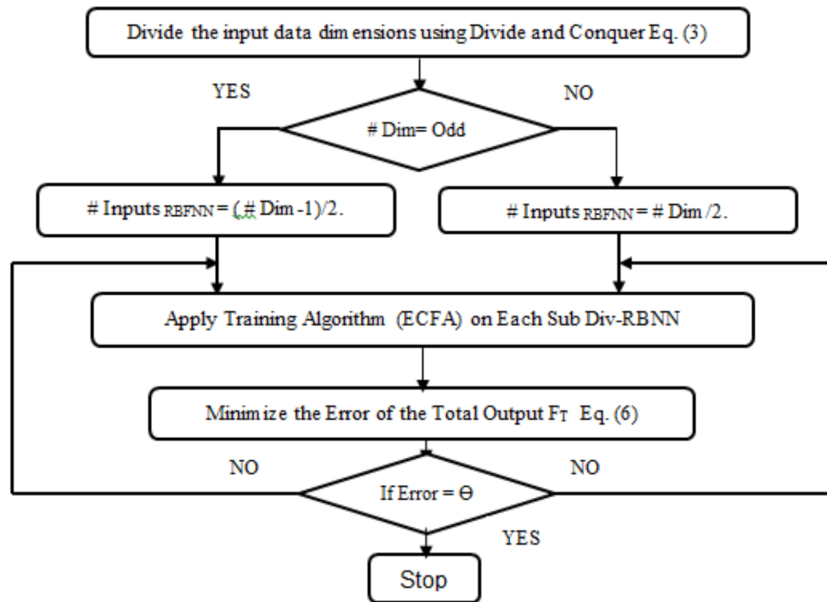


Figure 3. The proposed hybrid Div-RBFNN procedure.

Below is an illustration of the process of calculating the weight  $w_m^d$  :

$$w_m^d = G Y, \tag{7}$$

where  $G$  is the pseudoinverse matrix of the activation matrix  $\varphi_m^d$ .

In this way, the proposed hybrid method Div-RBFNN selects the suitable topology and optimizes the learning parameters of the RBFNN. Notably, some small subproblems (approximation problems with one dimension) may not be applicable with respect to the basic concepts of the proposed hybrid Div-RBFNN method; thus, the solution for multidimensional problems will be feasible. The Div-RBFNN topology is a powerful concept that can lead to a wide variety of applications in complex function approximation problems.

### 3. Numerical simulation examples

To demonstrate the efficiency of the proposed hybrid Div-RBFNN method on modeling complex function approximation problems, we proposed two different numerical functions (even/odd) with 5000 normalized data points for each data dimension generated randomly [8]. The proposed hybrid Div-RBFNN system was simulated in MATLAB 7.1 under Windows 7 with processor i5, 2.3 GHz, and 4 GB RAM to show three types of results: the topology of each Div-RBFNN, the number of executing parameters in each sub-Div-RBFNN compared with the number of executing parameters of a traditional RBFNN, and the approximation results of the validity of the proposed hybrid topology (test approximation error) derived from samples of complex input/output data, compared with approximation results of a traditional RBFNN that receives all the input data variables.

The results were obtained in 5 executions: {# of RBF in each sub Div-RBFNN} , the set of neurons used in each sub-Div-RBFNN. Par. #s is the number of executing parameters. A drop in this number assumedly reduces the execution time. The training data is supposed to be 70% of the input data in each dimension, and  $RMSE_{Test}$  is the root mean squared error of the test data, which is supposed to account for 30% of the input data.

**3.1. Even numerical examples,  $F_{\text{even}}(\mathbf{X})$**

The first hypothetical example involves 4 possible input data variables to be distributed by using a divide-and-conquer algorithm. Each variable consists of 5000 normalized input data points defined on the interval [0,1] and generated randomly from the proposed complex function shown below.

$$f_{\text{even}}(x) = \sin(\pi x_1 \cdot x_2) + 2(x_3 - 0.5)^2 + 10x_4 \cdot x_1, x_2, x_3, x_4 \in [0, 1], \tag{8}$$

This proposed function with even number of input data produces 2 different topologies: topology A with 1 input variable conveyed individually to 4 sub-Div-RBFNNs, and topology B with 2 input variables directed simultaneously to 2 sub-Div-RBFNNs. In all cases, the two proposed topologies produce fewer executing parameters, as shown in Table 2. The smallest number of executing parameters appears in the single input variables for each sub-Div-RBFNN. The proposed hybrid Div-RBFNN selects the best topology depending on the value of the approximation error. Figure 4 compares different topologies of the Div-RBFNN and traditional RBFNN.

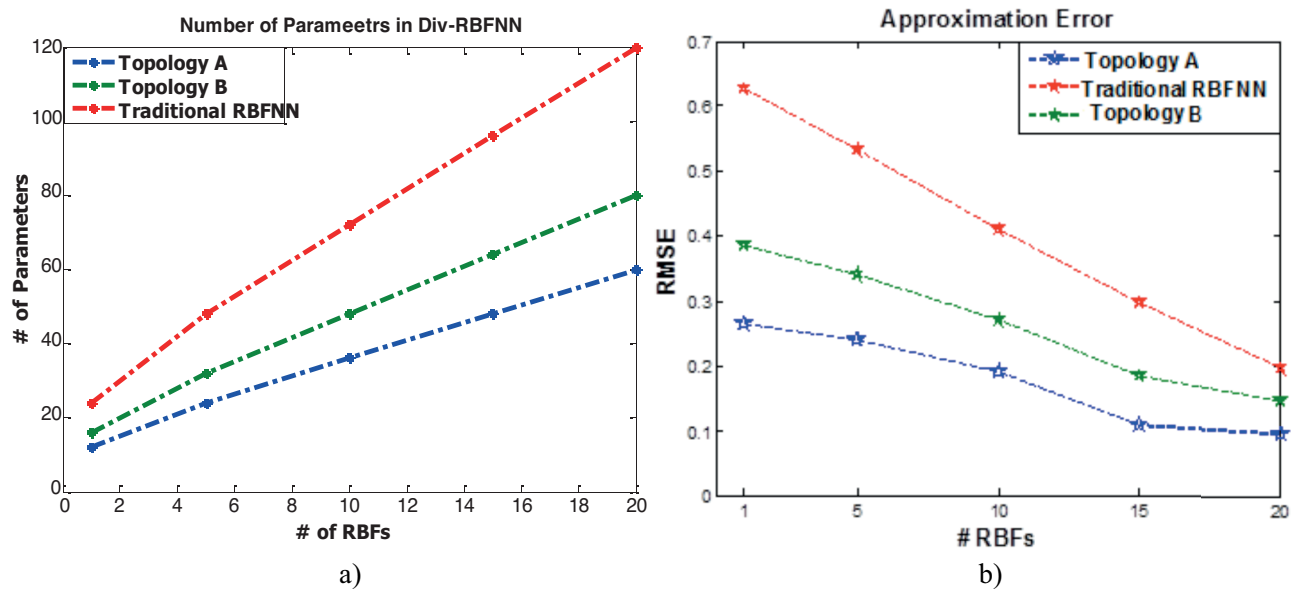


Figure 4. a) Parameter numbers in even Div-RBFNN, b) approximation error in even Div-RBFNN.

**3.2. Odd numerical examples,  $F_{\text{odd}}(\mathbf{X})$**

The second demonstration example consists of 5 possible input data variables to be distributed by using a divide-and-conquer algorithm. Each variable comprises 5000 normalized input data points defined on the interval [0,1] and generated randomly from the proposed complex function illustrated in the following equation.

$$f_{\text{Odd}}(x) = \sin(2\pi x_1 \cdot x_2) \cdot x_3 + e^{(-2 \cdot x_4 \cdot x_5)} \cdot x_1, x_2, x_3, x_4, x_5 \in [0, 1], \tag{9}$$

This proposed function with odd number of input data generates 4 different topologies. Topology A involves 1 input variable conveyed individually to 5 sub-Div-RBFNNs. Topology B consists of 2 sub-Div-RBFNNs: the first receives 2 input data variables and the other receives 3 input data variables. Topology C is the same as B, but functions contrariwise. The different topologies of B and C are used to ensure the validity of the



approximation process when the input data change. In topology D, 3 Div-RBFNNs are produced, with 2 sub-Div-RBFNNs receiving 2 input data variables each, while the third receives 1 input data variable. In all cases, the four proposed topologies produce fewer executing parameters, as shown in Table 3. Obviously the smallest number of executing parameters appears in the single input variables for each sub-Div-RBFNN. The proposed hybrid Div-RBFNN selects the best topology depending on the value of the approximation error.

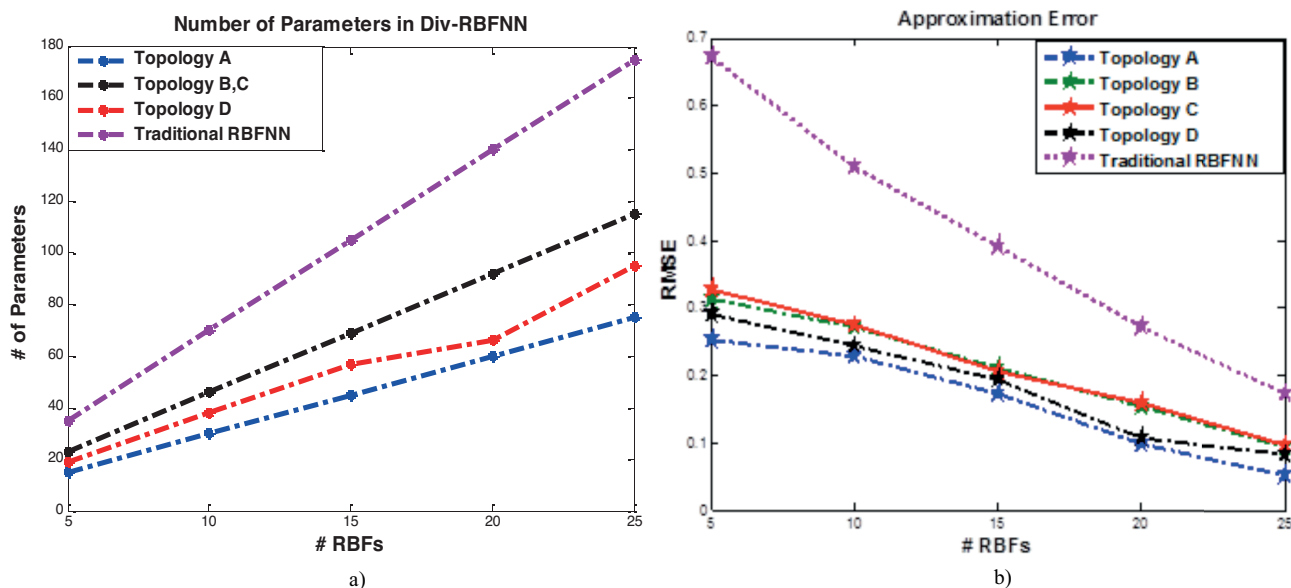
**Table 2.** Different topologies with their parameters in the even case and the approximation error for each one.

Div-RBFNN $f_{Even}(x)$						Traditional RBFNN		
A			B					
# of RBFs in Div-RBFNN	Par. #s	RMSE test	# of RBFs in Div-RBFNN	Par. #s	RMSE test	# of RBFs	Par. #s	RMSE test
{1 1 1 1}	12	0.265	{2 2}	16	0.385	4	24	0.627
{2 2 2 2}	24	0.240	{4 4}	32	0.341	8	48	0.534
{3 3 3 3}	36	0.192	{6 6}	48	0.272	12	72	0.411
{4 4 4 4}	48	0.109	{8 8}	64	0.185	16	96	0.297
{5 5 5 5}	60	0.096	{10 10}	80	0.147	20	120	0.198

**Table 3.** Different topologies with their parameters in the odd case and the approximation error for each one.

Div-RBFNN $f_{odd}(x)$					
A			B		
# of RBFs in Div-RBFNN	Par. #s	RMSE Test	# of RBFs in Div-RBFNN	Par. #s	RMSE test
{1 1 1 1 1}	15	0.252	{2 3}	23	0.312
{2 2 2 2 2}	30	0.227	{4 6}	46	0.271
{3 3 3 3 3}	45	0.172	{6 9}	69	0.210
{4 4 4 4 4}	60	0.098	{8 12}	92	0.154
{5 5 5 5 5}	75	0.051	{10 15}	115	0.093
C			D		
# of RBFs in Div-RBFNN	Par. #s	RMSE Test	# of RBFs in Div-RBFNN	Par. #s	RMSE test
{3 2}	23	0.327	{2 2 1}	19	0.291
{6 4}	46	0.274	{4 4 2}	38	0.244
{9 6}	69	0.206	{6 6 3}	57	0.194
{12 8}	92	0.158	{8 8 4}	66	0.106
{15 10}	115	0.097	{10 10 5}	95	0.083
Traditional RBFNN					
# RBFs	Par. #s		RMSE test		
5	35		0.671		
10	70		0.509		
15	105		0.391		
20	140		0.271		
25	175		0.172		

As Tables 2 and 3 clearly show, the success rates of the Div-RBFNN topology in our experiment were higher than the traditional RBFNN provided with the same clustering algorithm, ECFA [18]. Figure 5 draws a comparison between different topologies of Div-RBFNN and traditional RBFNN.



**Figure 5.** a) Parameter numbers in odd Div-RBFNN, b) approximation error in odd Div-RBFNN.

To illustrate, the proposed hybrid Div-RBFNN breaks the input data dimensions into smaller ones by using a divide-and-conquer algorithm, which generates fewer executing parameters, thus making the proposed hybrid Div-RBFNN more efficient in reducing the number of parameters and eventually the execution time.

The ECFA has been frequently used to solve function approximation problems. It basically looks for an optimal overall solution, which generates a sequence of improving approximate solutions for a class of problems, thus producing the best approximation result. The proposed Div-RBFNN method, however, divides the problem into simpler subproblems and groups all solutions. The methodology is best suited for modular complex data structures, with faster convergence compared to the traditional RBFNN.

#### 4. Conclusion

The basic drawback of complex approximation problems is that the increase in input data variables usually raises the number of executing parameters, even exponentially. This restricts the use of traditional modeling techniques owing to the execution time lag, which drives us to look for more effective solutions. In this paper, we proposed a hybrid Div-RBFNN topology based on breaking the problem input data variables into simpler problems. A complex problem model can be decomposed into a set of submodels; each represents different input data dimensions (odd/even). While each sub-Div-RBFNN model operates individually in the centers and radii parameters, the weights optimized rely on the total output. Eventually, the operation of the total system is enhanced in terms of reliability and fault tolerance. Div-RBFNN hybrid system output is derived from a combination of the products of the sub-Div-RBFNNs. As such, the approximation of the total (Div-RBFNNs) system is generally better than that of the traditional RBFNN. We provide a methodology that divides the input data variables of multidimensional function approximation problems and selects the most suitable topology for the proposed Div-RBFNN. The results produced by the Div-RBFNN outperform those derived by the traditional RBFNN in a number of executing parameters, suggesting shorter execution time and better approximation error.

## References

- [1] Vehtari A, Lampinen J. Bayesian Input Variable Selection Using Posterior Probabilities and Expected Utilities. Tech Report B31. Helsinki, Finland: Helsinki University of Technology, 2002.
- [2] Strass H, Wallner JP. Analyzing the computational complexity of abstract dialectical frameworks via approximation fix point theory. *Artif Intell* 2015; 226: 34-74.
- [3] Kainen P, Kurkov V, Sanguineti M. Dependence of computational models on input dimension: tractability of approximation and optimization tasks. *IEEE T Inform Theory* 2012; 58: 1203-1214.
- [4] Kůrková V. Dimension-independent rates of approximation by neural networks. In: Warwick K, Kárný M, editors. *Computer-Intensive Methods in Control and Signal Processing. The Curse of Dimensionality*. Boston, MA, USA: Birkhäuser, 1997. pp. 261-270.
- [5] Bengio S, Bengio Y. Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE T Neural Networ* 2000; 11: 550-557.
- [6] Bhagat S, Deodhare D. Divide and conquer strategies for MLP training. In: *IJCNN International Joint Conference on Neural Networks*; 16–21 July 2006; Vancouver, Canada. New York, NY, USA: IEEE. pp. 3415-3420.
- [7] Guo Q, Chen BW, Jiang F, Ji X, Kung SY. Efficient divide-and-conquer classification based on feature space decomposition. arXiv:1501.07584.
- [8] Awad M. Input variable selection using parallel processing of RBF neural Networks (PP-RBFNNs). *Int Arab J Inf Techn* 2010; 7: 6-13.
- [9] Ebrahimzadeh A, Khazae AA. An efficient technique for classification of electrocardiogram signals. *Adv Electr Comp Eng* 2009; 9: 89-93.
- [10] Ziehe A, Nolte G, Sander T, Muller KR, Curio G. A comparison of ICA based artifact reduction methods for MEG. In: *12th International Conference on Biomagnetism*; 13–17 August 2001; Helsinki, Finland. Helsinki, Finland: Helsinki University of Technology. pp. 895-899.
- [11] Ferrari S, Maggioni M, Borghese NA. Multiscale approximation with hierarchical radial basis functions networks. *IEEE T Neural Networ* 2004; 15: 178-188.
- [12] Pomares H, Rojas I, González J, Prieto A. structure identification in complete rule-based fuzzy systems. *IEEE T Fuzzy Syst* 2002; 10: 349-359.
- [13] Chen Y, Wang JZ. Kernel machines and additive fuzzy systems: classification and function approximation. In: *12th IEEE International Conference on Fuzzy Systems*; 25–28 March 2003; St Louis, MO, USA. New York, NY, USA: IEEE. pp. 789-795.
- [14] Cheung YM, Huang RB. A divide-and-conquer learning approach to radial basis function networks. *Neural Process Lett* 2005; 21: 189-206.
- [15] Noel S, Szu H. Multiple-resolution divide and conquer neural networks for large-scale TSP-like energy minimization problems. In: *IEEE International Conference on Neural Networks*; 12 June 1997; Houston, TX, USA. New York, NY, USA: IEEE. pp. 1278-1283.
- [16] Pomares H, Rojas I, Awad M, Valenzuela O. An enhanced clustering function approximation technique for a radial basis function neural networks. *Math Comput Model* 2012; 55: 286-302.
- [17] Dias FM, Antunes A, Vieira J, Mota A. A sliding window solution for the on-line implementation of the Levenberg–Marquardt algorithm. *Eng Appl Artif Intell* 2006; 19: 1-7.
- [18] Awad M, Pomares H, Rojas F, Herrera LJ, González J, Guillén A. Approximating I/O data using radial basis functions: a new clustering-based approach. In: *8th International Work-Conference on Artificial Neural Networks*; 5–7 June 2005; Barcelona, Spain. Berlin, Germany: Springer-Verlag. pp. 289-296.
- [19] Kumar RH, Kumar BV, Karthik K, Chand JL, Kumar CN. Performance analysis of singular value decomposition (SVD) and radial basis function (RBF) neural networks for epilepsy risk levels classifications from EEG signals. *Int J Soft Comput Eng* 2012; 2: 232-236.
- [20] Fulginei FR, Laudani A, Salvini A, Parodi M. Automatic and parallel optimized learning for neural networks performing MIMO applications. *Adv Electr Comp Eng* 2013; 13: 3-12.