Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 12 (9): 840-847, 2012
ISSN 1812-5654 / DOI: 10.3923/jas.2012.840.847

Learning Logic Programming in Radial Basis Function Network via Genetic Algorithm

Nawaf Hamadneh, Saratha Sathasivam, Surafel Luleseged Tilahun and Ong Hong Choon
School of Mathematical Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia

Abstract: Neural-symbolic systems are based on both logic programming and artificial neural networks. A neural network is a black box that clearly learns the internal relations of unknown systems. Radial Basis Function Neural Network (RBFNN) is a commonly-used type of feed forward neural network. Algorithms are used for learning the RBFNN in an adaptive procedure. Learning RBFNN indicate how the parameters (the output weights, the centers and the widths) should be incrementally adapted to improve a predefined performance measure, in this work, we embedded higher order logic programming in RBFNN. *k*-means cluster algorithm and Genetic Algorithm (GA) used in for training RBFNN.

Key words: Logic programming, Radial basis function neural network, Genetic algorithm, *k*-means cluster algorithm

INTRODUCTION

Neural-symbolic systems are Artificial Intelligence (AI) system which is a realization of symbolic processes within artificial neural networks. It is great to combine the robust artificial neural networks (Khatib and Alsadi, 2011; Ustun, 2007) and Logic programs. Designing an integrated system using Radial Basis Function Neural Network (RBFNN), can be benefited in utilizing the advantages of artificial neural networks and Logic programs. There have been numerous attempts for doing logic programming in artificial neural networks (Bader and Hitzler, 2004; Hitzler *et al*., 2004; Hölldobler and Kalinke, 1994; Seda, 2006) which used multilayer neural network.

The first step in the integration logic programming and artificial neural networks is by encoding logic programming within artificial neural networks. We presented a method of encoding higher order logic programming in Radial Basis Function neural network in by Hamadneh *et al.* (2012) (encoding method or EN method). It is obvious that RBFNN has been subjected to extensive research over recent years and have successfully been employed in various problem domains (Lowe, 1989; Moody and Darken, 1989). The idea of the RBFNN is to allocate each RBF neuron to respond to each of sub-spaces of a pattern class, formed by the clusters of training samples (Noman *et al*., 2009; Taghi *et al*., 2004; Xiaobin, 2009). Pursuant to that, learning at the hidden layer, is commonly configured as the problem of finding these clusters and their parameters by certain means of functional optimization. The name, RBFNN, comes from the fact that the radial basis functions which use in the hidden layer are radially symmetric. RBFNNs are very popular for function approximation, curve fitting, time series prediction and control and classification problems. The radial basis function network is different from other neural networks, possessing several distinctive features. Because of their universal approximation ability, more compact topology and faster learning speed, RBFNNs have attracted much attention and they have been widely applied in many science and engineering fields (Kang and Jin, 2010; Lu and Ye, 2007; Zayandehroodi *et al*., 2010). We also use the preference based genetic algorithm (Choon and Tilahun, 2011; Hasangholipour and Khodayar, 2010; Sinha and Chande, 2010) in order to adjust the output weights.

The objective of the study was to embedded higher order logic programming in RBFNN using *k*-means cluster algorithm and genetic algorithm.

PRELIMINARIES

Radial basis function neural network: RBFNN typically has three layers (Lowe, 1989; Moody and Darken, 1989; Rojas, 1996), namely, an input layer, a hidden layer with a non-linear RBF activation function, such as Gaussian function (Dhubkarya *et al*., 2010) and a linear output layer as shown in Fig. 1. It is a special class of multilayer feed-forward network (Zihe *et al*., 2011). The hidden layer neurons receive the input information, followed by certain decomposition, extraction and transformation steps to generate the output information. The following equation is the Gauss-radial basis function that we use in RBFNN (Dhubkarya *et al*., 2010):

Corresponding Author: Nawaf Hamadneh, School of Mathematical Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia

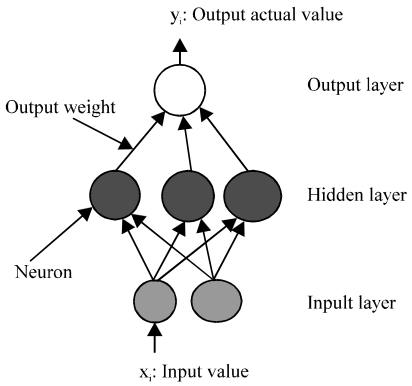

Fig. 1: Structure of a RBFNN

$$\varphi_i(x) = e^{-\left(\frac{\|x - c_i\|^2}{2\sigma_i^2}\right)} \quad (1)$$

where, φ_i is the radial basis function in hidden neuron i. $x \epsilon R$ is the input value. c_i and σ^2_i are the center and the width of the hidden neuron i, respectively.

It is worth mentioning that Eq. 1 is the radial basis function used in the hidden neurons in RBFNN without using constant input weights that which linked between input neurons and hidden neurons.

Each output unit implements a linear combination of this Radial Basis Function. From the point of view of function approximation, the hidden units provide a set of function that constitutes a basis set for representing input patterns in the space spanned by the hidden units.

A training process is used to determine the weights, the centers and the width. The weights are computed by applying an optimization algorithm for minimizing a suitable error function E. There are three different ways for training (Dhubkarya *et al*., 2010; Lampariello and Sciandrone, 2001) which are as follows:

- **No-training:** In this simplest case, all the parameters including the centers, the widths and the output weights, are calculated and fixed (no training is required). This paradigm does not have any practical value, because the number of the centers should be equal to the number of training samples (Dhubkarya *et al*., 2010)
- **Half-training:** In this case the hidden layer parameters (the centers and the widths) are calculated and fixed and only output layer weights are adjusted through
- **Full-training:** This paradigm requires the training of all parameters including the centers, the widths and the output weights

Logic programming: A logic program (Lloyd, 1984) is a set of axioms, clauses, or rules which in turn consist of literals, i.e., atoms and negated atoms only (negation is denoted by ¬). Note that, logic programming is activated by an initial goal statement. It is worth mentioning that Logical knowledge representation is symbolic. A higher order logic program (Lloyd, 1984) consists of a set of logic clauses each of the forms:

$$A_1 \vee A_2 \vee ... \vee A_n \leftarrow B_1 \wedge B_2 \wedge ...B_m \quad (2)$$

The clause (2) can be simplified to the following form:

$$\vee_{i=1}^{n} A_i \leftarrow \wedge_{j=1}^{m} B_j \quad (3)$$

where, the arrow may be read "if". The symbols $\vee$ and $\wedge$ read "or and" , respectively. $A_i \forall i$ and $B_j \forall i$ are literals.

Clauses can be either represented in Disjunctive Normal Form (DNF) or Conjunctive Normal Form (CNF) which is widely been used to represent clauses. Conjunction Normal Form (CNF) is a method of standardizing and normalizing logic programming. A logic programming P is CNF if and only if $P = F_1 \wedge F_2 \wedge F_3$ where F_i, i = 1,2,3 is a clause. Disjunction Normal Form (DNF) is a method of standardizing and normalizing each clause, for example, clause (2) can be written using DNF as follows:

$$\vee_{i=1}^{n} A_i \quad \vee_{j=1}^{m} \neg B_j \quad (4)$$

A clause which contains one atom in the head and literals in their body is called horn clause. Therefore, the general form of horn clauses is:

$$A_1 \leftarrow \wedge_{j=1}^{m} B_j \quad (5)$$

We allow m = 0, by an abuse of notation; in this case, the clause is called a unit clause or a fact clause.

EMBEDDING THE CLAUSES IN RADIAL BASIS FUNCTION NEURAL NETWORK

The primary end in this work is embedding higher order logic programming in RBFNN through ELPN method and genetic algorithm. Furthermore, we show, firstly, how to embed a clause in RBFNN through ELPN method, by using binary input neurons in RBFNN, where 0 refers to false and 1 refers to be true.

Consider that there are clauses in the form:

$$A_1 \vee A_2 \vee ... \vee A_k \leftarrow \neg B_1 \wedge \neg B_2 \wedge ... \wedge \neg B_x \wedge B_{x+1} \wedge B_{x+2} \wedge ... \wedge B_N \quad (6)$$

where, k, x, $N \epsilon \mathbb{N}$.

Table 1: The training set of the clause (10)

Input values	Output target values
4	1
3	1
2	1
1	1
0	1
-1	0

The clause (1) can be written using CNF as follows:

$$A_1 \vee A_2 \vee ... \vee A_k \vee B_1 \vee B_2 \vee ... \vee B_x \vee \neg B_{x+1} \vee \neg B_{x+2} \vee \neg ... \vee \neg B_N \quad (7)$$

The RBFNN which represents any clause consists of one input neuron and also one output neuron. As a rule, we will embed the clause (6) in RBFNN. Firstly, we convert the clause (6) to CNF form which become the clause (7). Accordingly, the input values in RBFNN be in the form:

$$A_1 + A_2 + ... + A_k + B_1 + B_2 + ... + B_x - B_{x+1} - B_{x+2} - ... - B_N = (\sum_{i=1}^{k} A_i + \sum_{i=1}^{x} B_i) - \sum_{i=x+1}^{N} B_i \quad (8)$$

The input data in Eq. 8 are given by Khatib and Alsadi (2011) Ustun (2007). Therefore, the output data of Eq. 8 be the following closed interval:

$$m \in [K + X, -(N - X)], m \in \mathbb{Z} \quad (9)$$

We summarize the steps of embedding the clause (6) in RBFNN using EN method by the following steps:

- **Step 1:** Convert the clause (6) into CNF form
- **Step 2:** We use one input neuron in RBFNN which represents the clause (6)
- **Step 3:** The number of hidden neurons is dependent on the method which is used in learning RBFNN. Moreover, half-training paradigm or full-training paradigm will reduce the number of hidden neurons
- **Step 4:** Eq.1 is the radial basis function which use in hidden neurons in RBFNN
- **Step 5:** The output data form of the input neuron in RBFNN is represents in Eq. 8. So the output data of the input neurons were represented by the closed interval (9)

We will discuss the clause (10) (Table 1) as a simple example of embedding the clauses in RBFNN:

$$A_1, A_2, A_3 \leftarrow \neg B_1, B_2 \quad (10)$$

We summarize the steps of embedding the clause (10) in RBFNN as follows:

- **Step 1:** Convert the clause (10) into CNF form. Therefore, the clause (10) be in the following form:

$$A_1 \vee A_2 \vee A_3 \vee B_1 \vee \neg B_2 \quad (11)$$

- **Step 2:** The RBFNN which represents the clause (10) consists of one input neuron and one output neuron
- **Step 3:** The number of hidden neurons is dependent on the method which uses in learning RBFNN. Moreover, half-training paradigm or full-training paradigm will reduce the number of hidden neurons. If we use the no-training paradigm to training RBFNN which represents the clause (10), we need to use six hidden neurons. Since, the input value be in the following form:

$$x = (A_1 + A_2 + A_3 + B_1) - B_2 \quad (12)$$

where, x is the output value of the input neuron

The digital values of the literals in Eq. 12 are {0,1}. Accordingly, the input value (x) in RBFNN which represents the clause (10), by using Eq. 12, is illustrated in the set (13).

$$x \in \{-1,0,1,2,3,4\} \quad (13)$$

- **Step 4:** Equation 1 is the radial basis function which use in hidden neurons in RBFNN which represents the clause (10)

EMBEDDING LOGIC PROGRAMMING IN RADIAL BASIS FUNCTION NETWORK

The following is a logic programming which will serve as an example for embedding logic programming in RBFNN:

$$\begin{aligned} C &\leftarrow \wedge \\ D &\leftarrow B \wedge \\ A &\leftarrow B, C, D \end{aligned} \quad (14)$$

The logic programming (14) consists of three clauses, so there are three output neurons and three input neuron. The number of the hidden neurons is dependent on the method used in learning RBFNN.

Note that, the ways we have chosen for training the logic programming (6) are no-training paradigm, as shown in Fig. 2. The logic programming (14) needs to be written in the following form which is necessary for selecting the input values:

$$C \wedge (D \vee \neg B) \wedge (A \vee \neg B \vee \neg C \vee \neg D) \quad (15)$$

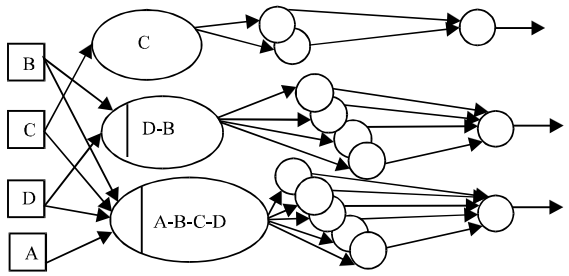

Fig. 2: Structure of RBFNN which represents the logic programming (14)

The output date form for each input neuron in RBFNN which represents the logic programming (15) is described in Fig. 2. Note that, the number of the hidden neurons in Fig. 1 is according to no-training method (Dhubkarya *et al*., 2010; Lampariello and Sciandrone, 2001).

TRAINING LOGIC PROGRAMMING IN RADIAL BASIS FUNCTION NETWORK

Training methods can be applied to the RBFNN parameters in order to improve the performance of the network. Training process (Asadollahi-Baboli, 2011; Lampariello and Sciandrone, 2001; Solaimani, 2009; Taghi *et al*., 2004) is used to determine the output weights, the centers and the widths. The training set for clause (11) is shown in Table 2. The RBFNN parameters can be determined by minimizing an error function that measures the degree of success. Noteworthy, as we mentioned earlier in section 2.1, there are three different paradigms for RBFNN (training RBFNN) which are: No-training, Half-training and Full-training.

The two phases of learning (Full-training) with RBFNN are supervised and unsupervised learning (Noman *et al*., 2009; Rojas, 1996). In this work we will use the full training paradigm to train RBFNN using *k*-means cluster algorithm and genetic algorithm.

***k*-means cluster algorithm:** *k*-means algorithm was developed by Idri *et al*. (2010), Looney (1997) and Zhang and Li (1996), is a popular clustering technique which is used in numerous applications. It is a multi-pass clustering algorithm. The *k*-means algorithm partitions a collection of N vectors into c clusters Z_i, i = 1,...,c. The aim is to find cluster centers by minimizing a distance function:

$$d(x_k,c_i) = |x_k-c_i| \quad (16)$$

where, c_i is the center of cluster z_i; $d(x_k,c_i)$ is the distance between ith center c_i and kth data point x_k.

After calculating the centers by using the *k*-means algorithm, we calculate the width for each hidden neuron with the variance σ^2. In the real numbers $\mathbb{R}$, the variance σ^2 for M elements be in the form:

$$\sigma_k^2 = \frac{1}{M_k}\sum_{x\in z_k}|x-c_k| \quad (17)$$

where, σ_k is the width of neuron k, M_k is the number of elements in the cluster Z_k.

The outline of the *k*-means algorithm (Negnevitsky, 2005; Zhang and Li, 1996) can be stated as follows:

- Define the number of the desired clusters Z, where each cluster is represented by a hidden neuron. So, the number of the clusters equal to the number of hidden neurons
- Initialize the centers c_i, i = 1,...,c. This is typically achieved by randomly selecting *c* points from among all of the data points
- Compute the distance between x_i and c_i, by using Eq. 16
- Assign each x_i to the closed cluster Z_i
- Upon averaging each cluster by Eq. 18, we obtain the new optimal centers:

$$c_i = \frac{1}{m_i}\sum_{v\in z_i} x \quad (18)$$

  where, c_i is the center of the cluster Z_i which has M elements
- Compute the objective function d given in Eq. 16. Stop if the clusters do not change
- If the elements in each cluster doesn't change terminate otherwise go to step 4

We will now calculate the centers of RBFNN which represent the clause (10) by using *k*-means algorithm. Then calculate the widths, as shown in Fig. 3. We have the input values {-1,0,1,2,3,4} with using three hidden neurons in RBFNN. Suppose that the initial values of the centers are 1, 3 and 4. Accordingly, we follow the steps which represent the *k*-means algorithm to determine the centers fixed as: $c_1 = 0$, $c_2 = 2.5$ and $c_3 = 4$.

By using Eq. 17, to calculate the widths, we get:

$$\sigma_1^2 = \frac{1}{3}(1+0+1) = \frac{2}{3},$$

$$\sigma_2^2 = \frac{1}{3}(0.5+0.5) = \frac{1}{3}$$

Table 2: Training set of the clause (10) in RBFNN with the radial basis function values using Eq. 1

i	Input values (x_i)	Output target value, (y_i)	$\phi_1(x_i)$	$\phi_2(x_i)$	$\phi_3(x_i)$
1	-1	0	0.184974	1.04482e-008	1.38879e-011
2	0	1	0.829205	8.47387e-005	1.12535e-007
3	1	1	0.829025	0.0342066	0.00012341
4	2	1	0.184974	0.687263	0.0183156
5	3	1	0.0092086	0.687263	0.367879
6	4	1	0.000102287	0.0342066	1

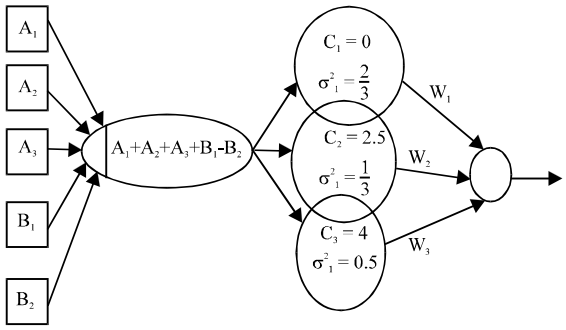

Fig. 3: RBFNN which represents the clause (10)

We have one element in third clause.

Suppose $\sigma_3^2 = 0.5$. Accordingly, the radial basis function values that we get by using Eq. 1 are found in Table 2.

Genetic algorithm: Genetic algorithm is one of the well known and widely used metaheuristic solution algorithms for optimization problems (Hasangholipour and Khodayar, 2010; Ustun, 2007). It has been used to solve many real problems (Wang and Wan, 2011). The algorithm imitates Darwin's theory of natural selection and survival of the fittest (Negnevitsky, 2005). In the algorithm an initial set of population will be generated encoded using 0's and 1's as a chromosome. Fitness for each solution member will be assigned depending on the objective function. For a maximization problem the fitness should be directly proportional to the functional value. In the algorithm there are two operators namely crossover operator and mutation operator.

The crossover operator is an operator which takes two solution members and produces two children by swapping some part of the chromosomes of the parents randomly. However, mutation operator is an operator which changes a randomly chosen chromosome from 0 to 1 or 1 to 0. The fittest member from both parent set and children will be selected for the next iteration. This process continues until termination a criterion fulfils.

The termination criteria could me maximum number of iterations or a specified small error (Negnevitsky, 2005).

The algorithm can be summarized as follows:

- Generate initial solutions and assign fitness for each solution member
- Choose parents for reproduction and mutation by giving high probability to the fittest
- Perform reproduction and mutation
- Select the fittest from parent and children and construct a new solution set
- If termination criteria fulfils stop else go to step 2

In the case of multiple objective functions, a dynamic weighted function can be generated as an aggregation objective function depending on the given preference or order of objective functions. Once the preference of the decision maker is known for each objective function, then that can be used as a weight for the objective functions in forming aggregated new objective function. The weight can be dynamic (Choon and Tilahun, 2011). In our case we set equal weight for each objective functions.

Applying GA to adjust the weights: We use a genetic algorithm by having a minimization multi-objective optimization problem with six objective functions to minimize the error of RBFNN which represent the clause (10). We set the upper bound and lower bound for the weights to be 2 and -2, respectively:

$$\min_{-2\le w_i\le 2}(f_1,f_2,f_3,f_4,f_5,f_6)$$

where:

$$f_i(w_1,w_2,w_3) = |w_1\phi_1(x_i) + w_2\phi_2(x_i) + w_3\phi_3(x_i) - y_i| \quad (19)$$

for $i\in\{1,2...,6\}$ which represents the absolute error between actual values and target values y_i.

The weight for each objective function is taken to be the same and a crisp number. Hence, the aggregated objective function, F, used to measure the fitness in the algorithm is given by:

$$F(w_1,w_2,w_3) = \sum_{i=1}^{6} f_i(w_1,w_2,w_3) \quad (20)$$

The probability of reproduction and mutation is taken to be 0.85 and 0.25, respectively. The total number of iteration is set 30 with number of initial population 40.

The best result after running the code is tabulated as follows:

Table 3: Values after running the genetic algorithm to find the weights

Input values (x_i)	Output target values, (y_i)	$\phi_1(x_i)$	$\phi_2(x_i)$	$\phi_3(x_i)$	Output values, $\sum_{i=1}^{3} w_i \phi_j(x_i)$
-1	0	0.184974	1.04482e-008	1.38879e-011	0.2168
0	1	0.829025	8.47387e-005	1.12535e-007	0.9718
1	1	0.829025	0.0342066	0.00012341	1.0034
2	1	0.184974	0.687263	0.0183156	0.8678
3	1	0.0092086	0.687263	0.367879	1.0001
4	1	0.000102287	0.0342066	1	0.9993

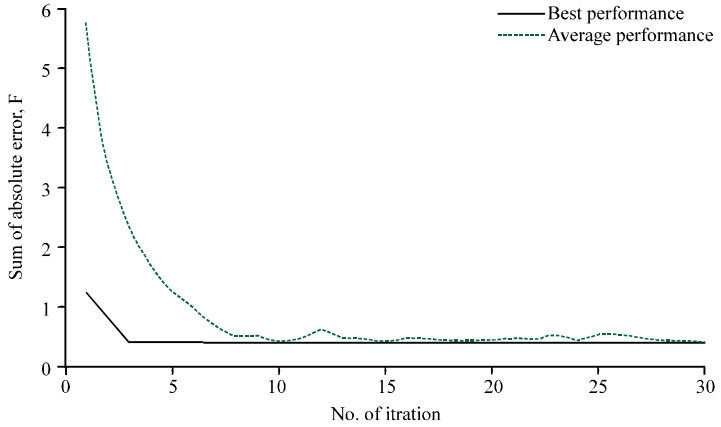

Fig. 4: The average and best performance of the algorithm in terms of sum of absolute errors

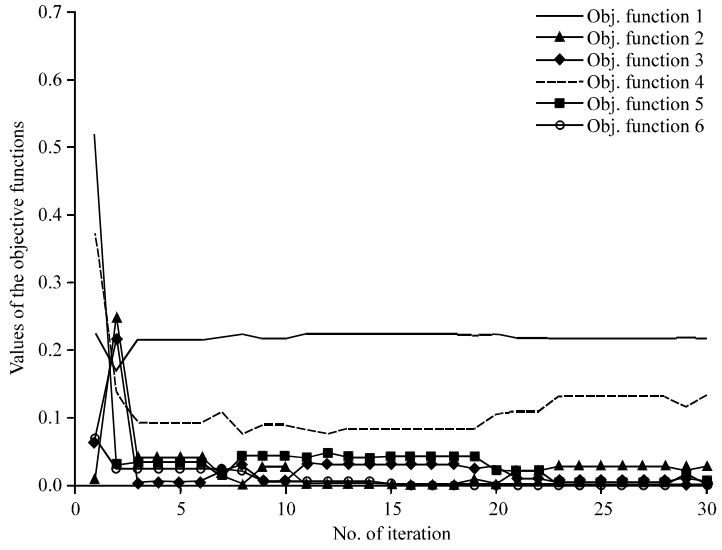

Fig. 5: The value of the objective functions (absolute errors) with the iteration of the algorithm

- The output weights of w_i, i = 1,2,3 are 1.1717, 0.9215 and 0.9676, respectively, with sum of absolute error of 0.3814
- The performance of the algorithm in terms of the best and average performance is given below, in Fig. 4 and 5. Note that, the actual output values were illustrated in Table 3

CONCLUSION

This study is an extension of Hamadneh *et al.* (2012) which use embedding higher order logic programming in RBFNN. Similarly, as in study of Hamadneh *et al.* (2012), we embedded higher order logic programming in RBFNN by using EN method. We trained the RBFNN for Table 1,

by using Genetic algorithm with *k*-means algorithm to improve the performance of the network. We used the *k*-means algorithm to find the centers and the widths of RBFNN and Genetic algorithm is used to adjust the output weighs. Accordingly, the sum of absolute error is 0.3814.

ACKNOWLEDGMENT

This research is party financed by Universiti Sains Malaysia's short term grant number: 304/PMATHS/6311126.

REFERENCES

Asadollahi-Baboli, M., 2011. Effect of weight updates functions in QSAR/QSPR Modeling using artificial neural networks. J. Artif. Intell., 4: 257-268.

Bader, S. and P. Hitzler, 2004. Logic programs, iterated function systems and recurrent radial basis function networks. J. Applied Logic, 2: 273-300.

Choon, O.H. and S.L. Tilahun, 2011. Integration fuzzy preference in genetic algorithm to solve multiobjective optimization problems. Far East Math. Sci., 55: 165-179.

Dhubkarya, D.C., D. Nagariya and R. Kapoor, 2010. Implementation of a radial basis function using VHDL. Global J. Comput. Sci. Technol., 10: 16-19.

Hamadneh, N., S. Sathasivam and O.H. Choon, 2012. Higher order logic programming in radial basis function neural network. Applied Math. Sci., 6: 115-127.

Hasangholipour, T. and F. Khodayar, 2010. A novel optimized neural network model for cost estimation using genetic algorithm. J. Applied Sci., 10: 512-516.

Hitzler, P., S. Holldobler and A.K. Seda, 2004. Logic programs and connectionist networks. J. Applied Logic, 2: 245-272.

Holldobler, S., Y. Kalinke, F.W. Ki, F. Informatik and T. Dresden, 1994. Towards a massively parallel computational model for logic programming. Proceedings of the ECCAI Workshop on Combining Symbolic and Connectionist Processing, August 9, 1994, Amsterdam, The Netherlands, pp: 68-77.

Idri, A., A. Zakrani and A. Zahi, 2010. Design of radial basis function neural networks for software effort estimation. Int. J. Comput. Sci. Issues, 7: 11-17.

Kang, P. and Z. Jin, 2010. Neural network sliding mode based current decoupled control for induction motor drive. Inform. Technol. J., 9: 1440-1448.

Khatib, T. and S. Al-Sadi, 2011. Modeling of wind speed for palestine using artificial neural network. J. Applied Sci., 11: 2634-2639.

Lampariello, F. and M. Sciandrone, 2001. Efficient training of RBF neural networks for pattern recognition. IEEE Trans. Neural Networks, 12: 1235-1242.

Lloyd, J.W., 1984. Foundations of Logic Programming. Springer-Verlag, Berlin, Germany.

Looney, C.G., 1997. Pattern Recognition Using Neural Networks: Theory and Algorithms for Engineers and Scientists. Oxford University Press, Oxford, UK., ISBN-13: 9780195079203, Pages: 458.

Lowe, D., 1989. Adaptive radial basis function nonlinearities and the problem of generalisation. Proceedings of the 1st IEE International Conference on Artificial Neural Networks, October 16-18, 1989, London, UK., pp: 171-175.

Lu, Y. and M. Ye, 2007. Oracle model based on RBF neural networks for automated software testing. Inform. Technol. J., 6: 469-474.

Moody, J. and C.J. Darken, 1989. Fast learning in networks of locally tuned processing units. Neural Comp., 2: 281-294.

Negnevitsky, M., 2005. Artificial Intelligence: A Guide to Intelligent Systems: Accelerated Learning in Multilayer Neural Network. 2nd Edn., Addison Wesley, Harlow, England, ISBN-13: 9780321204660, pp: 185-188.

Noman, S., S.M. Shamsuddin and A.E. Hassanien, 2009. Hybrid Learning Enhancement of RBF Network with Particle Swarm Optimization. In: Foundations of Computational Intelligence Volume 1: Learning and Approximation, Springer, Berlin, Germany, ISBN-13: 9783642010811, pp: 381-397.

Rojas, R., 1996. Neural Networks. Springer-Verlag, Berlin, Germany.

Seda, A., 2006. On the integration of connectionist and logic-based systems. Electron. Notes Theor. Comput. Sci., 161: 109-130.

Sinha, M. and S.V. Chande, 2010. Query optimization using genetic algorithms. Res. J. Inform. Technol., 2: 139-144.

Solaimani, K., 2009. A study of rainfall forecasting models based on artificial neural network. Asian J. Applied Sci., 2: 486-498.

Taghi, M., V. Baghmisheh and N. Pavesic, 2004. Training RBF networks with selective backpropagation. Neurocomputing, 62: 39-64.

Ustun, S.V., 2007. GA-based optimization of PI speed controller coefficients for ANN-modelled vector controlled induction motor. J. Applied Sci., 7: 4001-4006.

Wang, P. and P. Wan, 2011. The genetic algorithm to management measures of information security systems. Int. J. Phys. Sci., 6: 2934-2938.

Xiaobin, L., 2009. RBF neural network optimized by particle swarm optimization for forecasting urban traffic flow. Proceedings of the 3rd International Symposium on Intelligent Information Technology Application, November 21-22, 2009, University of Technology, Nanchang, China, pp: 124-127.

Zayandehroodi, H., A. Mohamed, H. Shareef and M. Mohammadjafari, 2010. Automated fault location in a power system with distributed generations using radial basis function neural networks. J. Applied Sci., 10: 3032-3041.

Zhang, Y.M. and X.R. Li, 1996. Hybrid training of RBF networks with application to nonlinear systems identification. Proceedings of the 35th Conference on Decision and Control, December 11-13, 1996, Kobe, Japan, pp: 937-942.

Zihe, G., G. Qing and N. Zhenyu, 2011. A distributed routing algorithm with traffic prediction in LEO satellite networks. Inform. Technol. J., 10: 285-292.