Applied Mathematical Sciences, Vol. 6, 2012, no. 3, 115 - 127

Higher Order Logic Programming in Radial Basis Function Neural Network

Nawaf Hamadneh, Saratha Sathasivam and Ong Hong Choon

School of Mathematical Sciences
Universiti Sains Malaysia
11800 USM, Penang, Malaysia
nwwaf977@gmail.com
saratha@cs.usm.my
hcong@cs.usm.my

Abstract

Intelligent systems are yielded from integration of a logic programming and connectionist systems. Radial basis function neural network is a commonly-used type of feedforward networks. In this paper, we proposed a method for connectionist model generation using Radial Basis Function neural network to encode higher order logic programming. We encode each clause from the clauses of a logic programming in separate networks, which are then reduced to a single network.

Keywords: Higher Order Logic Programming, Radial Basis Function Neural Network

1 Introduction

Artificial intelligence or AI systems are based on both logic programming and artificial neural networks[1, 2]. It is worth mentioning. One of AI goals is the creation of agents with human like intelligence. Neural-symbolic systems is Artificial intelligence system which is realization of symbolic processes within artificial neural networks. We take into account that logic programming and artificial neural networks both have different advantages and disadvantages. Logic programs are highly recursive and well understood from the perspective of knowledge representation. The success of artificial neural networks lies in the fact that they can be trained using raw data, and in some problem domains the generalization from the raw data made during the learning process turns out to be highly adequate for the problem at hand, even if the training data contains some noise. Designing an integrated system using feed-forward neural network such as Radial Basis Function neural network can benefit to their

advantages. There have been numerous attempts to doing logic programming in artificial neural networks. Steffen Holldobler and Yvonne Kalinke[3] proposed Propositional Core Method. The background knowledge of this method is represented as a neural symbolic integration cycle, within feed forward neural network.

Main challenges for the integration of logic programming and artificial neural network is in encoding symbolic knowledge or logic programming within such systems [1, 3, 4]. We present a new method of encoding higher order logic programming in Radial Basis Function neural network. The Radial Basis Function neural network or RBFNN has been subjected to extensive research over recent years and have successfully been employed in various problem domains (e.g. [5, 6]). The idea of a 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[7, 8, 9]. 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 basis functions in the hidden layer neurons are radially symmetric. A layer is a vector of units. 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, more compact topology and faster learning speed, RBFNNs have attracted much attention, and they have been widely applied in many science and engineering fields.

The purpose of this paper is twofold. First, we will give an overview of RBFNN and logic programming respectively in section two. We will then discuss in detail a new model using RBFNN on learning higher order logic programming. After that, in section four we look into some advantages about this model follow up by summary and discussion related to this work.

2 Preliminaries

2.1 Radial Basis Function Neural Network

RBFNN [5, 10] typically has three layers: namely an input layer, a hidden layer with a non-linear RBF activation functions and a linear output layer. It is a special class of multilayer feed-forward network, as shown in Figure1. The hidden layer neurons receive the input information, followed by certain decomposition, extraction, and transformation steps to generate the output information.

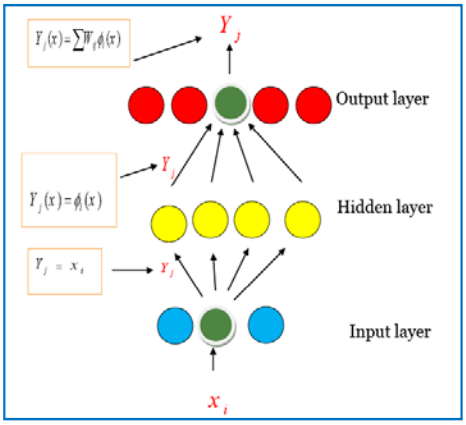

Figure 1: Structure of a RBF network

There are different types of neurons in the hidden layer and the output layer in RBFNN. Each hidden unit employs a Radial Basis Function, such as a Gaussian function, as the activation function. 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. The learning of the RBFNN requires the determination of the RBF centers, RBF widths and the weights. There are a variety of learning algorithms for the RBFNN. The basic one employs hybrid learning[7]. It estimates RBF centers and its widths by using an unsupervised clustering algorithm, followed by a supervised algorithm to determine the connection weights between the hidden layer and the output layer.

An approximation RBF neural network by a combination of Gaussian functions [11, 12] is of the form

$$\phi_i(x) = e^{-(\frac{\|x-c_i\|^2}{\sigma_i^2})} \tag{1}$$

where $i = 1, \ldots, l$, and l is the number of hidden neurons, c_i is the center of ith RBF hidden unit, which is a vector whose dimension is equal to the number of inputs to the neuron i, σ_i is the width of the receptive field in the input space from neuron i. This implies that ϕ_i has an appreciable value

only when the distance $\|x - c_i\|$ is smaller than the width σ_i. Meanwhile $\|.\|$ indicates the Euclidean norm on the input space.

2.2 Logic Programming

A logic program 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 $\neg$). Logic programming is activated by an initial goal statement. It is worth mentioning that Logical knowledge representation is symbolic. A higher order logic program[13]consists of a set of logic clauses each of the forms:

$$A_1, A_2, \ldots, A_K \longleftarrow B_1, B_2, \ldots, B_N, where K, N \in \mathbb{N}. \tag{2}$$

where the arrow may be read "if" and the commas "and". Note that, the program clauses are always satisfiable and solutions are guaranteed.

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 logical formulas. All logical formulas can be converted into CNF, a formula P is CNF if and only if $P = F_1 \wedge F_2 \wedge F_3$ where each F_i is a disjunction of literals(clause). Disjunction Normal Form (DNF) is a method of standardizing and normalizing each clause, for example, the clause (2) can be written using DNF as

$$A_1 \bigvee A_2 \bigvee A_3 \bigvee \ldots \bigvee A_K \bigvee \neg B_1 \bigvee \neg B_2 \bigvee \ldots \bigvee \neg B_N, where K, N \in \mathbb{N} \tag{3}$$

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:

$$A \longleftarrow B_1, B_2, \ldots, B_N, where N \in \mathbb{N}. \tag{4}$$

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

A logic program :

$$C \longleftarrow \bigwedge D \longleftarrow B \bigwedge A \longleftarrow B, C, D \tag{5}$$

which consists of three horn clauses. The logic programming (5) can be represented by combination of DNF and CNF as follows

$$(C) \bigwedge (D \bigvee \neg B) \bigwedge (A \bigvee \neg B \bigvee \neg C \bigvee \neg D) \tag{6}$$

Table 1: Input values and output target values of a clauses in RBFNN

Claus	Input values form	A	B	Input values	Output target values
$A \bigvee B$	$A + B$	0	1	1	1
		1	1	2	1
		1	0	1	1
		0	0	0	0
$A \bigvee \neg B$	$A - B$	0	1	-1	0
		1	1	0	1
		1	0	1	1
		0	0	0	1

3 Logic Programming in Radial Basis Function Neural Network

In the following, we established a new model for encoding higher order logic programming in RBFNN, with attention to the Radial Basis Function which is used the hidden layer, mentioned in equation1. We have benefited from the property in writing clauses using Disjunctive Normal Form (DNF) for the form of input values in RBFNN, as shown in Table1. In addition, the output target in Table1 equals1 for the satisfying values of A and B, and 0 for unsatisfying values of A and B. We described the new model in two subsections. Firstly, we used clauses and horn clauses, without any negated atoms in their bodes, for encoding higher order logic programming in RBFNN. Secondly, we used clauses and horn clauses with negated atoms.

3.1 Embedding clauses, contains only atoms in their bodies, in Radial Basis Neural Network

The primary end in this subsection is embedding higher order logic programming, that consists of clauses or horn clauses, includes only atoms in their bodies, in RBFNN. We will show firstly, how to embed the horn causes (4) in RBFNN, by using binary input neurons, where 0 refers to false and 1 refers to be true. Note that, the clause (4) consists of N atoms in their body without any negated atoms. We used logic programming (5) to clarify the encoding higher order logic programming, who consists of horn clauses, in RBFNN. After that, we generalize the model for embedding the clause (2) in RBFNN.

Summarizing steps of embedding the horn clause (4), in RBFNN, which are described in Figure2, is as follows

[1)]

We have $N + 1$ literals in the clause (4). Accordingly, the numbers of neurons

in RBFNN are: $N+1$ neurons in hidden layer, one neuron in input layer and one neuron in output layer. Note that, N is natural number and greater than zero in the new model.

Clause (4) can be written on the form $A \bigvee \neg B_1 \bigvee \neg B_2 \bigvee \ldots \bigvee \neg B_N$. Accordingly, the input values of RBFNN which represented clause (4) is of the form $A-(B_1+B_2+\ldots+B_N)$, for all literals $A, B_1, B_2, \ldots, B_N$. Each literal takes randomly a binary value (0 or 1). So, the input values are $-N, -N+1, -N+2, -N+3, \ldots, 1$. Note that, N is the number of atoms in the body of clause (4).

To calculate and fixed the centers of the hidden neurons, we take it as a subset of input values as follows $-N, -N+2, -N+3, \ldots, 1$.

To calculate and fixed the width (σ), we used the following formula $\sigma = d_{max}/\sqrt{\text{number of hidden neurons}}$,
where d_{max} is the maximum distance between the centers. As a result, $\sigma = 1$ for all centers.

The target output value equals1 for satisfactory input values, and 0 for unsatisfactory input values. Accordingly, the target output value equals1 for the following input values $-N+1, -N+2, -N+3, \ldots, 1$, and 0 if and only if the input value equal $-N$.

After calculating and fixing the centers and the width, and knowing the output targets for each input value, we calculate the weights between the hidden neurons and output neuron, with adding threshold value, which needs to be computed.

The values of $\{ A, B_1, B_2, \ldots, B_N\}$ are satisfying if and only if
$|\textit{actual output value} - 1| < \textit{tolerance value}$.

We want to encode logic programming (5) in RBFNN, as an example for the new model, as shown in Figure4. To achieve this, we embed each clause of logic programming (5) in a separate network, as shown in Figure3. The following is the steps of encoding logic programming (5) in RBFNN. [1)]

We have 2 literals in the clause $D \longleftarrow B$ and 4 literals the clause $A \longleftarrow B, C, D$. So, by using step1 in the new model to calculate the number of hidden neurons, the number of hidden neurons of the clauses $D \longleftarrow B$ and $A \longleftarrow B, C, D$ are 2 and 4 respectively.

The clause $C \leftarrow$ is a fact clause. We use two hidden neurons in any fact clause. Note that, fact clause is the special case in the new mode.

By using the second step in the new model, the input values form for each RBFNN, that each one of them represents the clause $C \leftarrow$, $D \longleftarrow B$ and $A \longleftarrow B, C, D$, are C, D-B and A-(B+C+D) respectively. For clarification, the input values in RBFNN for the clauses $C \leftarrow$, $D \longleftarrow B$ and $A \longleftarrow B, C, D$, are {1,0},{1,0,-1}and {1,0,-1,-2,-3} respectively.

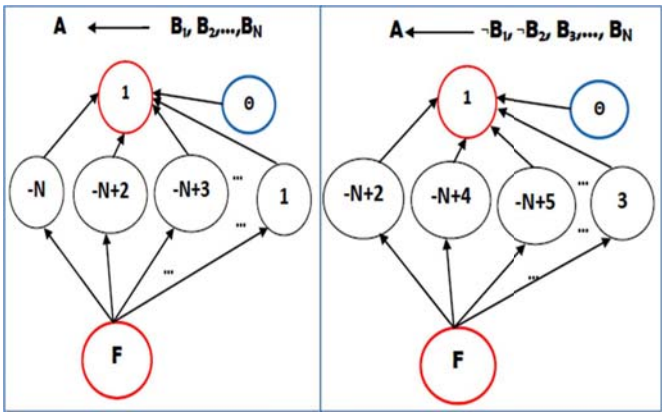

Figure 2: Structure of horn clauses in RBF Neural Network without any negated atoms(left) and horn clauses in RBF Neural Network with negated atoms (right)

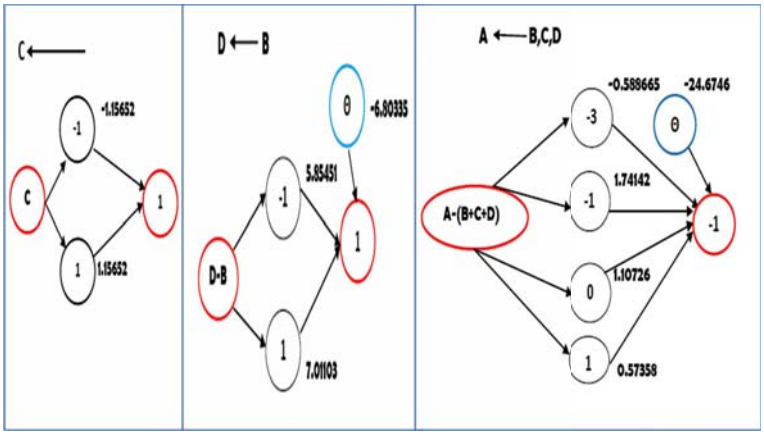

Figure 3: Structure of the clauses $C \longleftarrow, D \longleftarrow B,\ A \longleftarrow B, C, D$ in RBF Neural Network

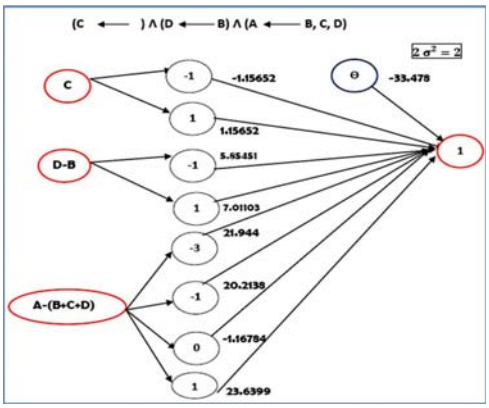

Figure 4: Structure of the logic program$\{(A \bigvee \neg B \bigvee \neg C \bigvee \neg D) \bigwedge (D \bigvee \neg C) \bigwedge (C)$ in RBF Neural Network

We have 1 atom in the body of the clause $D \longleftarrow B$ and 3 atoms in the body of the clause $A \longleftarrow B, C, D$, so in accordance to the third step in the new model, the centers in each RBFNN are {-1,0} and {-3,-1,0,1} respectively, with adding threshold value.

The centers in RBFNN which represent the special case clause $C \longleftarrow$ are {1,-1}, without adding threshold value.

As we said in step five in the new model, the target output value equals1 for satisfactory input values, and 0 for unsatisfactory input values. The satisfactory input values for the clauses $C \longleftarrow$, $D \longleftarrow B$ and $A \longleftarrow B, C, D$ are {1},{0,1}and{-2,-1,0,1} respectively.

After calculating the centers and the width for the hidden neurons in each clause, and locating the target output for each input value, it's easy to estimate the weights between hidden neurons and out put neuron in each RBFNN. Note that, the centers, the widths and the weights are the parameters of RBFNNs.

The final step is to build one RBFNN, which represents the logic programming(4), as shown in Figure4. So, we collect the networks that each one representing a clause in logic programming(4). The target output value for satisfactory input values, in each clause, is equal1. So, the total target output in RBFNN which corresponding the logic programming(4) equal3, so we subtract 2 from the threshold value, to become the target output equal1.

This new model can be generalized to include the higher order logic programming in the form of clause(2), which can be written in the form of clause(3). As shown in Figure5, we embedded the clause(2) in RBFNN as the following steps: [1)]

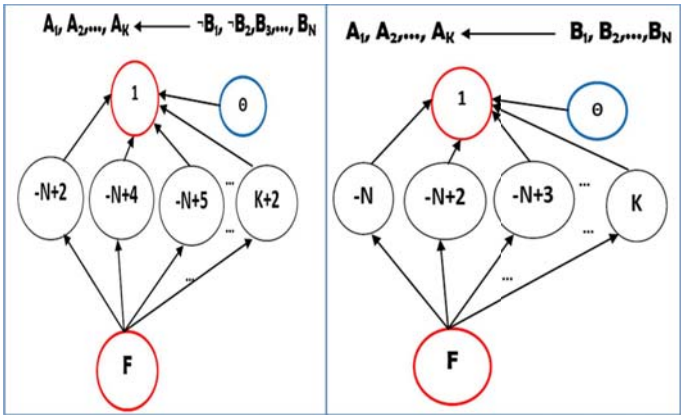

Figure 5: Structure of clauses in RBF Neural Network without any negated atoms(left) and clauses in RBF Neural Network with two negated atoms (right)

We have N+K literals the clause(2). So, the numbers of neurons in RBFNN are: N+K neurons in the hidden layer, one neuron in input layer and one neuron in output layer.

According clause(3), the input values of clause(2) in RBFNN is of the form $A_1 + A_2 + \ldots + A_K - B_1 - B_2 - \ldots - B_N = \sum_{i=1}^{K} A_i - \sum_{j=1}^{N} B_j$, for all literals. Each literal takes randomly a binary value (0 or 1). So, the input values are $\{-N, -N+1, -N+2, -N+3, \ldots, K\}$.

To calculate and fixed the centers of the hidden neurons, we take it as a subset of input values as follows $\{-N, -N+2, -N+3, -N+4, \ldots, K\}$.

To calculate and fixed the width (σ), we used the following formula
$\sigma = d_{max}/\sqrt{\text{number of hidden neurons}}$,
where d_{max} is the maximum distance between the centers. As a result, $\sigma = 1$ for all centers.

The target output value equals1 for satisfactory input values, and 0 for unsatisfactory input values. Accordingly, the target output value equals1 for the following input values $-N+1, -N+2, -N+3, \ldots, K$, and 0 if and only if the input value equal $-N$.

After calculating and fixing the centers and the width, and knowing the output targets for each input value, we calculate the weights between the hidden neurons and output neuron, with added threshold value, which needs to be computed.

The input values in RBFNN are satisfied if and only if
$|actual\ output\ value - 1| < tolerance\ value$.

3.2 Embedding clauses, contains negated atoms in their bodies, in Radial Basis Neural Network

This subsection is an extension to the subsection3.1, by taking, firstly, the second case for horn clauses, which consists negated atoms in their body. We have the horn clause

$$A \longleftarrow \neg B_1, \neg B_2, B_3, \ldots, B_N, where N \in \mathbb{N} \tag{7}$$

which consists of two negated atoms.

To embed the clause(7) in RBFNN, as shown in Figure2, we follow the following steps [1)]

We have $N+1$ literals in the clause(7). Accordingly, the numbers of neurons in RBFNN are: $N+1$ neurons in hidden layer, one neuron in the input layer and one neuron in output layer. Note that, N is natural number and greater than zero in the new model.

The horn clause(7) can be represented by DNF as follows

$$A \bigvee B_1 \bigvee B_2 \bigvee \neg B_3 \bigvee \ldots \bigvee \neg B_N, where N \in \mathbb{N} \tag{8}$$

Accordingly, the input values of RBFNN which is represented clause(7) is of the form $A + B_1 + B_2 - (B_3 + \ldots + B_N)$, for all literals $A, B_1, B_2, \ldots, B_N$. Each literal takes randomly a binary value (0 or 1). In addition, we have $N-2$ negated atoms and 3 atoms in clause(8). So, the input values in RBFNN are $\{-N+2, -N+3, -N+4, -N+5, \ldots, 3\ \}$.

To calculate and fixed the centers of the hidden neurons, we take it as a subset of input values as follows $\{-N+2, -N+4, -N+5, \ldots, 3\}$.

To calculate and fixed the width (σ), we used the following formula
$\sigma = d_{max}/\sqrt{\text{number of hidden neurons}}$,
where d_{max} is the maximum distance between the centers. As a result, $\sigma = 1$ for all centers.

The target output value equals1 for satisfactory input values, and 0 for unsatisfactory input values. Accordingly, the target output value equals1 for the following input values $\{-N+3, -N+4, -N+5, \ldots, 3\}$, and 0 if and only if the input value equal $-N+2$.

After calculating the centers and the width for the hidden neurons the clause, and locating the target output for each input value, it's easy to estimate the weights between hidden neurons and output neuron in each RBFNN.

The values of $\{A, B_1, B_2, \ldots, B_N\}$ are satisfied if and only if
$|actual\ output\ value - 1| < tolerance\ value$.

We will now embed the clause

$$A \longleftarrow \neg B, \neg C, D, M \tag{9}$$

as an example of horn clause which contains negated atoms in their body, as follows: [1)]

We have 5 literals in the clause(9). Accordingly, the numbers of neurons in RBFNN are: 5 neurons in hidden layer, one neuron in input layer and one neuron in output layer.

Rewrite the clause (9) to the following form

$$A \bigwedge B \bigwedge C \bigwedge \neg D \bigwedge \neg M \tag{10}$$

So, the input values are of the form A+B+C-D-M. We have 3 atoms and 2 negated atoms in clause(10). Therefore, the input values in RBFNN are {-2, -1, 0, 1, 2, 3}.

To calculate and fixed the centers of the hidden neurons, we take it as a subset of input values as follows { -2, 0, 1, 2, 3} with width equal one.

The target output value equals1 for satisfactory input values, and 0 for unsatisfactory input values. Accordingly, the target output value equals1 for the following input values $\{-1, 0, 1, \ldots, 3\}$, and 0 if and only if the input value equal -2.

The input values are satisfied if and only if
$|actual\ output\ value - 1| < tolerance\ value$.

Embedding clauses, which consist of two negated atoms in the body, that in the form

$$A_1, A_2, \ldots, A_K \longleftarrow \neg B_1, \neg B_2, B_3 \ldots, B_N \tag{11}$$

in RBFNN, as shown in Figure5, is as follows [1)]

we have N+K literals in clause(11). So, we have N+K neurons in hidden layer, one neuron in input layer and one neuron in output layer.

According to the clause(11) which can be written, by using DNF, in the form
$A_1 \bigvee A_2, \ldots \bigvee A_K \bigvee B_1 \bigvee B_2 \bigvee \neg B_3 \bigvee \ldots \bigvee \neg B_N$
the input values in RBFNN are of the form $A_1 + A_2 + \ldots + A_K + B_1 +$

$B_2 - B_3 - \ldots - B_N = \sum_{i=1}^{K} A_i + B_1 + B_2 - \sum_{j=3}^{N} B_j$. Therefore, the input values in RBFNN are $\{ -N+2, -N+3, \ldots, K+2\}$, with the centers $\{ -N+2, -N+4, -N+5 \ldots, K+2\}$ and the width, which equal1 for all hidden neurons.

The target output value equals1 for satisfactory input values, and 0 for unsatisfactory input values. Accordingly, the target output value equals1 for the following input values $\{ -N+3, -N+4 \ldots, K+2\}$, and 0 if and only if the input value equal -N+2. Note that, there are two negated atoms in the body of clause(11).

More specifically, if there are X negated atoms in the body of a clause which consists of N literals in the body and K atoms in the head, then the input values in corresponding networks are { -N+X,-N+X+1,-N+X+2, ... ,K+X}, with the centers { -N+X,-N+X+2, ... ,K+X} and the width, which equal1 for all hidden neurons.

4 Conclusion

In this paper, We have proposed a new model for embedding higher order logic programming into Radial Basis Function Neural Network. To explain the new model, we firstly build RBFNN for each to clauses that a higher order logic programming consists from them. After that, reduced the networks to a single RBFNN, which will represent the higher order logic programming.

Before generalizing the new model to embed whether the general form of the clauses, contain or do not contain negated atoms in their bodies, we explained the embedding horn clauses in RBFNN, by using the new model.

Acknowledgements

This research is party financed by short term grant by Universiti Sains Malaysia (304/PMATHS/631006)

References

[1] P. Hitzler, S. Hölldobler and A. Seda, Logic programs and connectionist networks, *Journal of Applied Logic*. **2**(2004), 245 - 272 .

[2] S. Sathasivam, *Learning Rule Performance Comparison in Hopfield Network*, *American Journal of Scientific Research*. (2009), 15 - 22.

[3] S. Hölldobler and Y.Kalinke, Towards a massively parallel computational model for logic programming., *Proc. ECAI Workshop on Combining Symbolic and Connectionist Processing,ECCAI.* (1994), 68 -77 .

[4] A. Seda, On the Integration of Connectionist and Logic-Based Systems,*Electronic Notes in Theoretical Computer Science.* **161** (2006), 109 - 130 .

[5] J. Moody and C.J. Darken, Fast learning in networks of locally tuned processing units, *Neural Computation*, **1** (1989), 281 - 294.

[6] D. Lowe, Adaptive radial basis function nonlinearities, and the problem of generalisation, *First IEE International Conference*, (1989), 171 - 175.

[7] S. Noman, S. M. Shamsuddin, and A. Hassanien, *Hybrid Learning Enhancement of RBF Network with Particle Swarm Optimization*, Springer-Verlag Berlin Heidelberg, 2009.

[8] M. Taghi, V. Baghmisheh and N. Pavesic, Training RBF networks with selective back propagation,*Neurocomputing.* **62** (2004), 39 - 64 .

[9] L. Xiaobin, *RBF Neural Network Optimized by Particle Swarm Optimization for Forecasting Urban Traffic Flow*, *Third International Symposium on Intelligent Information Technology Application.* (2009), 124 - 127.

[10] R.Rojas, Neural Networks, Springer-Verlag, Berlin, 1996.

[11] D. C. Dhubkarya, D. Nagariya and R. Kapoor, Implementation of a Radial Basis Function Using VHDL, *Global Journal of Computer Science and Technology.* **10** (2010), 16 - 19 .

[12] A.dri, A. Zakrani and A. Zahi, IDesign of Radial Basis Function Neural Networks for Software Effort Estimation, *IJCSI*, **7** (2010), 11 - 17.

[13] J. W. Lloyd, *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1984.

Received: May, 2011