International Journal of Computational and Electronics Aspects in Engineering
Vol. 1, Issue. 1, Dec 2014, Revised in March 2020, pp 16-24
https://doi.org/10.26706/ijceae.1.1.20141204

Comparing Logic Programming in Radial Basis Function Neural Network (RBFNN) and Hopfield Neural Network

Mamman Mamuda[1]
maanty123@gmail.com

Saratha Sathasivam[2]
saratha@usm.my

School of Mathematical Sciences, Universiti Sains Malaysia, 11800 USM, Penang, Malaysia

***Abstract*— Neural network is a black box that clearly learns the internal relations of unknown systems. Neural-symbolic systems are based on both logic programming and artificial neural networks. Radial basis function neural network (RBFNN) and Hopfield neural network are the two well-known and commonly used types of feed forward and feedback networks. This study gives an overview of how logic programming is been carried out on both networks as well as the comparison of doing logic programming on both radial basis neural network and Hopfield neural network.**

Index terms— Radial basis function neural network, Hopfield network, Logic programming

I. INTRODUCTION

Neural-symbolic structures are artificial intelligence system that can realize the symbolic processes within artificial neural networks. An artificial neural network which is also known as connectionist systems [1] has received precise attention because of their capacity to evaluate complex non-linear data set. Neural networks are artificial intelligence structures that include some mathematical and graphical models. Neural networks or connectionist architectures provide an alternative computational paradigm [2], and can be seen as a step towards the understanding of intelligence. It departs from the traditional way of serial processing and instead is based on distributed processing via connections between simple elements. This is motivated by biology, and offers new and alternative ways of computation [3]. The models are used to show the structure of the artificial neural network. There are different types of artificial neural networks and all execute principally the same functions [1], their applications are distinctive cases of vector mapping. They comprise of an enormous number of modest processing elements called neurons, cells, unites or nodes [4]. It can be shown that prepositional logic programming can be done on the model of a single neuron [3]. Neurons are connected with each other by uninterrupted communication links, each with a related weight. Two main topologies in neural network are considered according to their connectivity. The first one is the Radial Basis Function Neural Network (RBFNN) which is a feed forward network, and the second one is the Hopfield Neural Network which is the feedback neural network. Neurons in the Radial Basis Function neural network belonging to the same layer receive inputs from neurons of the preceding layer and send their values only to neurons of the resulting layer [5]. In Hopfield

Research Paper
First Online on – 30 Dec 2014, Revised on – 30 March 2020

Cite this article – Mamman Mamuda and Saratha Sathasivam, "Comparing Logic Programming in Radial Basis Function Neural Network (RBFNN) and Hopfield Neural Network", *International Journal of Computational and Electronics Aspects in Engineering*, RAME Publishers, vol. 1, issue 1, pp. 16-24, 2014, Revised in 2020.
https://doi.org/10.26706/ijceae.1.1.20141204

neural network; neurons belonging to the same layer send their output to neurons of the resulting and the preceding layers. Logic programming and artificial neural networks both have different advantages and disadvantages. Logic programs are extremely recursive and well understood from the angle of knowledge representation [6]. Artificial neural networks succeed in that, they can be trained using raw data, and sometimes the difficult domains that are generalize from the raw data made during the learning process turns out to be highly adequate for the problem at hand. The training consists of fine-tuning the weights on the interconnections in the network until the error which is the difference between the real output and the target output is small. The first phase in the combination of logic programming and artificial neural networks is by encoding logic programming within neural networks [5]. The objective of this study is to compare the method of doing logic programming in radial basis function neural network (RBFNN) and Hopfield neural network. The paper is therefore organized as follows. In section 2, we briefly explain the architecture of Radial basis function neural network. In section 3, the architecture of Hopfield neural network will also be explained. The outline of doing logic programming in both radial basis function neural network (RBFNN) and Hopfield neural network (HNN) will be in section 4 and 5 respectively. Section 6 contains the summary of the similarity and dissimilarity of doing logic programming in both RBFNN and HNN. We give some concluding remarks of this work in section 7.

II. Radial Basis Function Neural Network (RBFNN)

The idea of radial basis function neural network (RBFNN) is to assign each radial basis function (RBF) neuron to react to each of subspaces of a pattern class, formed by the clusters of training models [7, 8, 2]. Radial basis function neural network (RBFNN) typically has three layers [4, 9]: namely; an input layer, a hidden layer with a non- linear RBF activation functions and linear output layer. It is a special class of multilayer feed-forward network. The hidden layer neurons accept the input information, trailed by definite decomposition, extraction and transformation steps to produce the output information. A radial basis function neural network is a special type of neural network that uses a radial basis function as its activation function [10]. The numbers of neurons in the hidden layer of a radial basis function neural network are determining because it affects the complexity and generalizing the capability of the network. A layer is a vector of units. Radial basis function neural networks are very common for function approximation, curve fitting, time series prediction, and control of classification problems. Because of their complete approximation and more compact topology and fast learning speed, radial basis function neural network (RBFNN) have fascinated much attention and have been broadly useful in many science and engineering fields.

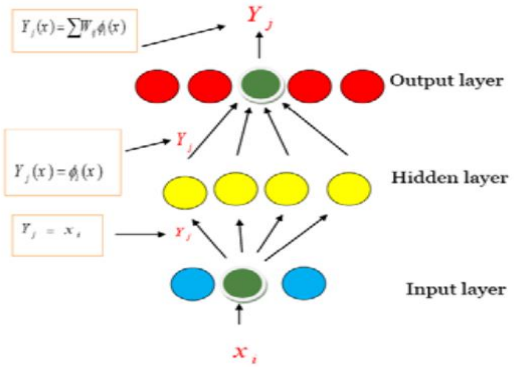

Figure1. Structure of RBFNN

Radial basis function neural network (RBFNN) sets the constant of the weight among the input layer and hidden layer and appraises the weight among the hidden layer and output layer. The weights between the hidden-to-output are usually assign to be 0 which stands for the connection to the output that is false and 1 which stands for the connection to the output that is true. The input layer connects the network to its environment that is made up of source neurons. The hidden layer neurons are linked with centers that decide the behavior and the structure of the network. The output layer which is the last layer of the RBFNN provides the reaction or response of the network to the activation pattern of the input layer that serve as a summation unit. The name

RBFNN arises from the point that the basis functions in the hidden layer neurons are radially symmetric. The radial basis function which is usually denoted by φ_i is usually computed by the i^{th} hidden neurons and is maximum when the input vector is near the center of that neuron. Many types of radial basis functions are considered, such as:

$$\phi_i(x) = e^{-\left(\frac{\| x - c_i \|^2}{\delta_i^{\,2}}\right)} \text{--------- Gaussian function} \quad (1)$$

$$\phi_i(x) = \frac{1}{1 + e^{-f_i(x)}} \text{ ----------Sigmoidal function} \quad (2)$$

$$\phi_i(x) = \frac{1 - \exp(-2z_j)}{1 + \exp(-2z_j)} \text{--Hyperbolic tangt. Function} \quad (3)$$

etc.

The radial basis function network can be approximated by the combination of any of the functions above [4]. For example, an approximation of radial basis function network by the combination of Gaussian function can be seen as follows:

$$y_i(x) = \sum_{i=1}^{N} wij\phi_i(x) \text{----------} (4)$$

Where,

$$\varphi_i(x) = \exp-\left(\frac{\| x - c_i \|^2}{2\delta_i^{\,2}}\right) \text{----------} (5)$$

where $i = 1,..., N.$

C_i is the center of the radial basis function in the hidden unit which is a vector whose dimension is equivalent to the number of inputs to the neuron *i*, δ_i is the width of the receptive field in the input space from neuron *i*. ||.|| signifies the Euclidian norm on the input space. φ_i Can have an appreciable value when the distance $\| x - c_i \|$ is smaller than the width δ_i.

In radial basis function neural network (RBFNN), it is very essential to determine the number of neurons in the hidden layer; this is because it touches the network complexity and generalizing the competence of the network. The centers in the hidden i.e. (the position of the center) affects the substantially performance of the network. It is therefore important to also determine the optimal locations of centers. Radial basis function neural network has two types of learning: supervised and unsupervised (hybrid learning). Learning in RBFNN are aim to determine a set of weights which minimizes the error, since a supervised learning such as back propagation learning algorithm has a problem of error convergence. The optimization of centers of each neuron in radial basis neural network can be employ by a training procedure, after which the weights between the hidden layer and the output layer must be selected appropriately. A bias value would be added with each output and are determined in the training procedure of the radial basis function neural network.

III. Hopfield Neural Network

The Hopfield neural network is one of the artificial neural networks that are propagated by john Hopfield [12] in 1982 based on the minimization of energy function. The architecture of Hopfield neural network consist of two dimensional connected neural networks in which the linking strengths between neurons- that are binary threshold neurons are determined based on the constraints and solution criteria of the optimization problem to be solved.

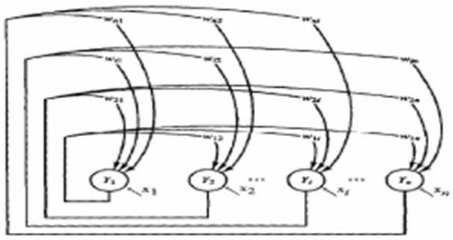

Figure 2. Structure of a discrete Hopfield network

Each unit is connected to all other units except itself which avoids a permanent feedback of its own state value. Hopfield neural network is a feedback neural network. A solution in Hopfield neural network is active after the network is relaxed and stretches to a stable state. The energy function in Hopfield network [13] in higher order is of the form:

$$\left.\begin{aligned} E &= \ldots - \frac{1}{3}\sum_i\sum_j\sum_k J^{(3)}_{ijk} s_i s_j s_k - \frac{1}{2}\sum_i\sum_j J^{(2)}_{ij} s_i s_j - \sum_i J_i s_i \\ h_i &= + \sum_i\sum_j\sum_k J^{(3)}_{jk} s_j s_k + \sum_i\sum_j J^{(2)}_{ij} s_j + \sum_i J_i \end{aligned}\right\} \quad (6)$$

The system in Hopfield model consists of N formal neurons each of which is defined by an Ising variable s_i that is called the state of neuron i, $s_i \in (1,-1)$ and $s_i = sign(h_i)$. It has an updating rule i.e. $s_i(t+1) = sign(h_i(t))$ and $J^{(2)}_{ij}$ is the synaptic strength from neuron i to j. The synaptic strengths are symmetric with zero diagonal [11] i.e.

$J^{(n)}_{ij\ldots k} = J^{(n)}_{ji\ldots k}$ and $J^{(n)}_{ii\ldots i} = J^{(n)}_{jj\ldots j} = 0.$

Hopfield network must have a learning rule that sets the connection weights between all pairs of neurons such as Hebbian learning and Wan Abdullah's learning method [15]. Every learning algorithm of perceptron's for Hopfield network can be turned into a learning method. The learning methods are used to calculate the changes in connection weights depending on the activities of the neurons. For example; the Hebbian learning method is given by:

$\Delta J^{(n)}_{i,j\ldots,m} = \lambda^{(n)} \sum_{k=1}^{m} s_i s_j \ldots s_m$ Where λ is the learning rate

$J_{i,j}{}^{(m)} \ldots, m = \lambda^{(n)} \sum_{k=1}^{m} s_i s_j \ldots s_m$ Where

$\lambda^{(n)} = \dfrac{1}{(n-1)!}$ in which time and learning thresholds can be taken into account. Different learning rules produce different basins of attraction [16].

IV. LOGIC PROGRAMMING IN RADIAL BASIS NEURAL NETWORK (RBFNN)

The objectives of doing logic programming in neural network is to find a set of interpretations i.e. the truth value of the atoms in the clauses that satisfy the clauses, i.e. which yields all the clauses to be true. We first start by embedding the horn clauses in RBFNN by using binary input neurons, where 0 refers to false and 1 refers to be true.

Considering the clause below:

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

Where $k, x, N \in \mathbb{N}$

The clause (7) is in radial basis function neural network with the binary input neurons where 0 refers to false and 1 refers to be true. The clause (7) can now be converted to a conjunctive normal form (CNF) which becomes:

$$A_1 \vee A_2 \vee \ldots \vee A_k \vee B_1 \vee B_2 \vee \ldots \vee B_x \vee \neg B_{x+1} \vee \neg B_{x+2} \vee \neg \ldots \vee \neg B_N \quad ---(8)$$

In radial basis function neural network; the input values of clause (8) is in the form:

$$\begin{aligned} &A_1 + A_2 + \ldots + A_k + B_1 + B_2 + \ldots + B_x - B_{x+1} - B_{x+2} - \ldots - B_N \\ &= \left(\sum_{i=1}^{k} A_i + \sum_{i=1}^{k} B_i \right) - \sum_{i=x+1}^{N} B_i \end{aligned} \quad (9)$$

Consider the horn clause below:

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

Where the commas means "and" and arrow means "if" The horn clause (10) can now be embedded in radial basis function neural network using the following steps:

Step I: Convert the horn clause (10) into conjunctive normal form (CNF) i.e.

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

Step II: The radial basis functions that represent the clause (11) consist of one input neuron and one output neuron.

Step III: Select a training pattern to train the RBFNN that represents the clause (11). This will determine the number of hidden neuron to be use. Since the number of hidden neurons is dependent on the method used in learning RBFNN. Three (3) different ways for training the RBFNN are considered. (i) No training (ii) Half training and (iii) Full training.

Step IV: Determine the input value of the clause (11). This will be in the 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 literal in (12) are (0, 1)

For instance, the input value (X) which represents the clause (10) in RBFNN by using Eq. 12 is as follows:

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

Having embedded the horn clauses in Radial basis function neural network, we now embed the logic programming in radial basis function neural network

Consider the below logic programming as an example;

$$\left.\begin{array}{l} A \leftarrow B, C, D \\ D \leftarrow B \\ C \leftarrow \end{array}\right\} \quad (14)$$

The logic programming (14) comprises of three clauses; therefore there will be three output neurons and three input neurons. Depending on the training pattern selected; we can re-write the logic programming (14); for instance selecting the no- training pattern, the logic programming can be written in the following form:

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

The number of the hidden neurons is dependent on the method used in learning RBFNN. We can now apply a training pattern to train the RBFNN parameters in other to increase the performance of the network. Training process is used to determine the output weight, the centers and the widths. The RBFNN parameters can be determine by minimizing an error function that measures the degree of achievement. For instance, if we take the Gaussian function as our activation function, i.e

$$\phi_{(x)} = e^{-}\left(\frac{\| x - c_i \|^2}{2\delta_i^2}\right) \quad (16)$$

Then the center and the width will be determined using a clustering algorithm. For example, using the K-means cluster algorithm, the collection of N vectors will be partition into C- clusters; $z_i, i = 1, ..., c$ with the aim of finding cluster centers by minimizing a distance function given by

$$d\left(x_k, c_i\right) = | x_k - c_i | \quad (17)$$

Where c_i is the center of cluster z_i; $d\left(x_k, c_i\right)$ is the distance between i^{th} center c and k^{th} data point x_k. After obtaining the center using the K-means algorithm, we then obtained the width for each hidden neuron with the variance δ^2.

Having obtained the center and the width of the network, we can now obtain the values of the radial basis function which is our activation function represented in (16).

After calculating and fixing the centers and the width, also after knowing the output targets for each of the input value, we then calculate the weights between the hidden neurons and output neurons with added threshold value which needs to be computed. The input values in radial basis function neural network (RBFNN) are satisfied if and only if:

|actual output value-1| < tolerance value

The following flowchart illustrates the process of doing logic programming in radial basis function neural network.

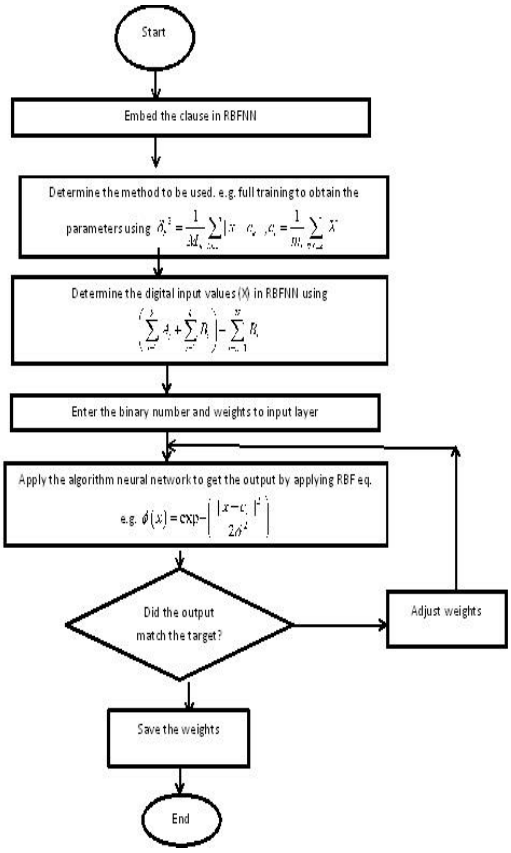

Figure 3 Flowchart of doing logic programming in RBFNN

V. LOGIC PROGRAMMING IN HOPFIELD NEURAL NETWORK

In this section, we discuss how to carryout logic programming in Hopfield neural network. In principle, logic programming in Hopfield neural network can be seen as a problem in combinatorial optimization. Many optimization problems, including those that are associated with intelligent behavior can easily be represented in Hopfield network with the aim of transforming the problem into variables such that the desired configuration corresponds to the minimization of the respective lyapunov function. The energy function can be thought of as a programming language for which the optimization problems can be transformed by applying the network dynamics into a solution method. Logic programming in Hopfield neural network is done by using the neurons to store the truth values of the atoms and writing a cost function which is minimized when all the clauses are satisfied. Logic clauses of the form $A \leftarrow B_1, B_2, ..., B_n$ are set of Horn clauses that can be used. The objective is to find the set(s) of interpretation i.e. truth values for the atoms in the clauses that would yield all the clauses to be true. For example, consider the following logic program:

$$\left.\begin{array}{l} A \leftarrow B, C \\ D \leftarrow B \\ C \leftarrow \end{array}\right\} \quad (18)$$

The logic program (18) has three clauses which can be translated as:

$A \vee \neg B \vee \neg C; D \vee \neg B$ and C. The task of the program is to look for interpretations of the atoms. In this case, atoms A, B, C and D makes up the model for the given logic program. It can be seen as a combinatorial optimization problem where the inconsistency of the lyapunov or energy function will be of the form:

$$E_p = \frac{1}{2}(1-s_A)\frac{1}{2}(1+s_B)\frac{1}{2}(1+s_C)+\frac{1}{2}(1-s_D)\frac{1}{2}(1+s_B)+\frac{1}{2}(1-s_C) \quad ---(19)$$

Where $s_{A,}s_{B,}s_C$ and s_D represent the truth values (true as 1) of A, B, C and D to be chosen as the cost function to be minimized. It can therefore be observed that the minimum value of E_p is 0, and has otherwise value proportional to the number of unsatisfied clauses. The energy function (19) when programmed onto a third order neural network using Wan Abdullah's method [15] produce the synaptic strength as:

For clause $A \leftarrow B, C;$

$$\left.\begin{array}{l} J_{[ABC]}^{(3)} = \frac{1}{16}; J_{[ABD]}^{(3)} = J_{[ACD]}^{(3)} = J_{[BCD]}^{(3)} = 0 \\ J_{[AB]}^{(2)} = J_{[AC]}^{(2)} = \frac{1}{8}; J_{[AD]}^{(2)} = J_{[BD]}^{(2)} = J_{[CD]}^{(2)} = 0; \\ J_{[BC]}^{(2)} = -\frac{1}{8} \\ J_{[A]}^{(1)} = \frac{1}{8}; J_{[B]}^{(1)} = J_{[C]}^{(1)} = -\frac{1}{8}; J_{[D]}^{(1)} = 0. \end{array}\right\} ---(20)$$

For clause $D \leftarrow B;$

$$\left.\begin{array}{l} J_{[ABC]}^{(3)} = J_{[ABD]}^{(3)} = J_{[ACD]}^{(3)} = J_{[BCD]}^{(3)} = 0; \\ J_{[AB]}^{(2)} = J_{[AC]}^{(2)} = J_{[AD]}^{(2)} = \\ J_{[BC]}^{(2)} = J_{[CD]}^{(2)} = 0; J_{[BD]}^{(2)} = \frac{1}{4} \\ J_{[A]}^{(1)} = J_{[C]}^{(1)} = 0; J_{[B]}^{(1)} = -\frac{1}{4}; J_{[D]}^{(1)} = \frac{1}{4}. \end{array}\right\} ---(21)$$

For clause $C \leftarrow$

$$\left.\begin{array}{l} J_{[ABC]}^{(3)} = J_{[ABD]}^{(3)} = J_{[ACD]}^{(3)} = J_{[BCD]}^{(3)} = J_{[AB]}^{(2)} = J_{[AC]}^{(2)} = \\ J_{[AD]}^{(2)} = J_{[BC]}^{(2)} = J_{[CD]}^{(2)} = J_{[BD]}^{(2)} = 0 \\ J_{[A]}^{(1)} = J_{[B]}^{(1)} = J_{[D]}^{(1)} = 0; J_{[C]}^{(1)} = \frac{1}{2} \end{array}\right\} ----(22)$$

To obtain logic program using Hopfield network as proposed by [12]; the following algorithm is to be carried out.

(i) Given a logic program; transform all the clauses in the logic program into a Boolean algebraic form.

(ii) Identify a neuron to each ground neuron.

(iii) Initialize all connections strength to zero.

(iv) Derive a cost function that is associated with the negation of the conjunction of all the clauses, such that ½(1+S_x) represents the logical value of a neuron corresponding to logical atom x. the value of S_x is define in such a way that it carries the values of 1 if x is true and -1 for x is false. The negation (x does not occur) is represented by ½(1-S_x); a conjunction logical connective is represented by multiplication where as a disjunction connective is represented by addition.

(v) Obtain the values of connection strength by comparing the cost function with the energy.

(vi) Let the neural network programmed with this connection strengths progress until minimum energy is reach.

The following flowchart shows how logic programming has been applied in Hopfield neural network

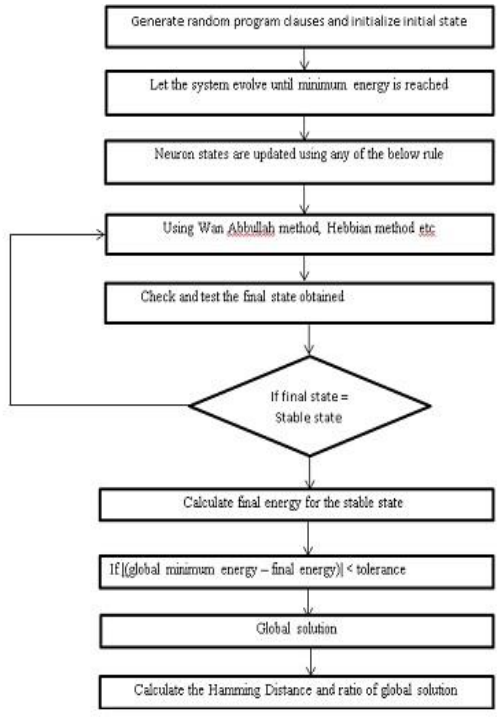

Figure 4. Flowchart of doing logic programming using Hopfield neural network

VI. Comparison Of Doing Logic Programming Between RBFNN And Hopfield Neural Network

In this section, we are going to compare doing logic programming in radial basis function neural network (RBFNN) and Hopfield neural network. We categorize this into similarities and dissimilarities of doing logic programming in both radial basis function neural network and Hopfield neural network.

A. Similarities

- Logic programming in both RBFNN and Hopfield neural network uses logic programs that consist of set of program clauses and are activated by initial goals statements.
- The clauses are always in conjunctive normal form (CNF) which contains one positive literal.
- They both have the same objectives, i.e finding a set of interpretation (truth value of the atoms in the clauses which satisfy the clauses; which yield all the clauses true).
- There processing units are either in binary states (1,-1) or bipolar states (1, 0).
- Both RBFNN and Hopfield network uses an algorithm in getting the output of the network.

B. Dissimilarities

- The set of program clauses in logic programming using RBFNN are determined by their digital values of the literals, where by in Hopfield neural network, the set of clauses in the logic program are transformed into a Boolean algebraic form.
- Doing logic programming in RBFFN, a training method i.e No training, half training or Full training have to be selected in training the parameters of the RBF from the hidden layer, where by logic programming in Hopfield neural network does not require that.
- In Hopfield neural network, logic programming can be done by deriving a cost function that is associated with the negation of the conjunction of all the clauses such that $\frac{1}{2}(1+S_x)$ represents the logical value of a neuron corresponding to logical atom X. the value of S_x is define in such a way that it carries the values of 1 if X is true and -1 if for X is false, and its negation is represented by $\frac{1}{2}(1-S_x)$. In doing logic

programming in RBFNN, any of the activation function used in the hidden neuron, for example, the Gaussian function, the parameters are to be first determined.

- Logic programming in RBFNN uses hybrid learning algorithm to train the network i.e. the parameters of the RBF i.e. the center and the width of the hidden units were obtained by unsupervised type of learning and the output weights were obtained by supervised learning, on the other hand, logic programming in Hopfield neural network uses only one type of learning, i.e. it uses an unsupervised type of learning in finding patterns in the input space in order to train the network.
- Weights calculation between the hidden neurons and output neurons in logic programming using RBFNN can be computed with an addition of a threshold value. In Hopfield network, synaptic strengths can be obtained for each of the clauses**.**

VII. CONCLUSION

This study explains the procedure of carrying out logic programming in both radial basis function neural network (RBFNN) and the Hopfield neural network. Clauses were obtained and embedded in both the networks. The algorithms of doing logic programming in both the networks were explained and compared. In doing logic programming in radial basis neural network, the parameters of the network i.e. (the center and the width) would first be determined. The procedure for determining the parameters were explained. The center and the width are very necessary condition for the convergence that connects the network to a stable state.

VIII. ACKNOWLEDGEMENT

This research was supported by FRGS grant (203/PMATHS/6711368) offered by Ministry of Higher Education and Universiti Sains Malaysia.

REFERENCES

[1] S. Sathasivam, N. Hamadneh, O. H. Choon, Comparing neural networks: Hopfield network and rbf network, Applied Mathematical Sciences 5 (2011) 3439–3452.

[2] W. W. Abdullah, The connectionist paradigm, Proceedings 1st National Computer Science Conference (1989) 95–111.

[3] W. W. Abdullah, Computationas with neural networks and neural-network-like structures, *Computational Techniques and Applications:CTAC, 87* (1988).

[4] J. Moody, C. J. Darken, Fast learning in networks of locally-tuned processing units, Neural computation 1 (1989) 281–294.

[5] N. Hamadneh, S. Sathasivam, O. H. Choon, Higher order logic programming in radial basis function neural network, Appl Math Sci 6 (2012) 115–127

[6] S. Sathasivam, Learning in the recurrent hopfield network, in: Computer Graphics, Imaging and Visualisation, 2008. CGIV'08. Fifth International Conference on, IEEE, pp. 323–328.

[7] S. Noman, S. M. Shamsuddin, A. E. Hassanien, Hybrid learning enhancement of rbf network with particle swarm optimization, in: Foundations of Computational, Intelligence Volume 1, Springer, 2009, pp. 381–397.

[8] M.-T. Vakil-Baghmisheh, N. Pavešić, Training rbf networks with selective backpropagation, Neurocomputing 62 (2004) 39–64.

[9] R. Rojas, Neural networks: a systematic introduction, Springer, 1996.

[10] D. S. Broomhead, D. Lowe, Radial basis functions, multi-variable functional interpolation and adaptive networks, Technical Report, DTIC Document, 1988

[11] A. Idri, A. Zakrani, A. Zahi, Design of radial basis function neural networks for software effort estimation, IJCSI International Journal of Computer Science Issues 7 (2010).

[12] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proceedings of the national academy of sciences 79 (1982) 2554–2558.

[13] S. Sathasivam, W. A. T. W. Abdullah, Logic learning in hopfield networks, arXiv preprint arXiv:0804.4075 (2008).

[14] S. Sathasivam, Neuro-symbolic performance comparison, in: Computer Engineering and Applications (ICCEA),

2010 Second International Conference on, volume 1, IEEE, pp. 3–5

[15] S. Sathasivam, W. A. T. W. Abdullah, Logic mining in neural network: reverse analysis method, Computing 91 (2011) 119–133.

[16] A. J. Storkey, R. Valabregue, The basins of attraction of a new hopfield learning rule, Neural Networks 12 (1999) 869–876.