Far East Journal of Mathematical Sciences (FJMS)
Volume 47, Number 2, 2010, Pages 185-200

NORMAL FORM REPRESENTATION IN DOING LOGIC PROGRAMMING IN HOPFIELD NETWORK

SARATHA SATHASIVAM

School of Mathematical Sciences
Universiti Sains Malaysia
11800 USM, Penang, Malaysia
e-mail: saratha@cs.usm.my

Abstract

The main purpose of doing logic programming in Hopfield network is to obtain suitable models for the corresponding logical clauses. In order to represent the logical clauses, the conjunctive normal form (CNF) and the disjunctive normal form (DNF) have been proposed. In this paper, we will illustrate the effectiveness of CNF and DNF in representing the logical clauses. We will also discuss the ways to convert DNF formulas to CNF formulas. The satisfiability aspects of Horn clauses in the form of CNF will be also discussed. We have proved that CNF representation is better than DNF in logical reasoning system. Computer simulation has also been carried out to verify the proposed theory.

I. Introduction

Logic program and neural networks are two important paradigms in artificial intelligence. Logic describes relationship among propositions. Consequently, logic must have descriptive symbolic tools to represent propositions. Representation of neural networks on the other hand is in non-symbolic form. Neural networks are massively parallel with self-learning capabilities. Neural networks are specified by the net topology, node characteristics, and training or learning rules. Thus, logic

---

2010 Mathematics Subject Classification: 68T27.

Keywords and phrases: logic programming, Hopfield, CNF, DNF.

Received August 10, 2010

programs and neural networks seem to be rather complementary. It would be desirable to integrate both approaches in order to combine their remarkable abilities [1]. Suitable methods for the extraction of knowledge from neural networks are therefore being sought within many ongoing research projects worldwide [1, 2]. Some other publications, for example, [3, 4], also underscore the importance of these two paradigms. By having a logical framework, knowledge reasoning techniques can be employed.

In 1982, Hopfield [2] proposed a fully connected neural network model of associative memory in which patterns can be stored by distributed among neurons, and we can retrieve one of the previously presented patterns from an example which is similar to, or a noisy version of it. The dynamical behavior of the neurons in neural network depends on synaptic strength between neurons. The specification of the synaptic weights is conventionally known as learning. Hopfield applied Hebbian learning rule [3] to determine weights. However, many alternative algorithms for learning and associative recalling have been proposed to improve the performance of the Hopfield networks [4, 5].

Wan Abdullah [6] proposed a method of doing logic program on a Hopfield network. Optimization of logical inconsistency is carried out by the network after the connection strengths are defined from the logic program; the network relaxes to neural states which are models (i.e., viable logical interpretations) for the corresponding logic program. Clauses can either be represented in Conjunctive Normal Form (CNF) or Disjunctive Normal Form (DNF). However, CNF is widely been used to represent clauses. In this paper, we will prove the equalities between the CNF and DNF conversion in carrying out logic program in Hopfield network. We will also discuss the advantages and disadvantages of both the representation of logical clauses. This work is an extension to our previous work [6].

II. Logic Programming on a Hopfield Network

In order to keep this paper self-contained, we briefly review the Little-Hopfield model. The Hopfield model is a standard model for associative memory. The Little dynamics is asynchronous, with each neuron updating their state deterministically. The system consists of N formal neurons, each of which is described by an Ising variable $S_i(t)\ (i = 1, 2, ..., N)$. Neurons are bipolar, $S_i \in \{-1, 1\}$, obeying the dynamics

$S_i \to \mathrm{sgn}(h_i)$, where the field $h_i = \sum_j J_{ij}^{(2)} V_j + J_i^{(1)}$, *i* and *j* running over all neurons *N*, $J_{ij}^{(2)}$ is the synaptic strength from neuron *j* to neuron *i*, and $-J_i$ is the threshold of neuron *i*.

Restricting the connections to be symmetric and zero-diagonal, $J_{ij}^{(2)} = J_{ji}^{(2)}$, $J_{ii}^{(2)} = 0$, allows one to write a Lyapunov or energy function,

$$E = -\frac{1}{2}\sum_i \sum_j J_{ij}^{(2)} S_i S_j - \sum_i J_i^{(1)} S_i \tag{1}$$

which monotone decreases with the dynamics.

The two-connection model can be generalized to include higher order connections. This modifies the "field" to be

$$h_i = \cdots + \sum_j \sum_k J_{ijk}^{(3)} S_j S_k + \sum_j J_{ij}^{(2)} S_j + J_i^{(1)}, \tag{2}$$

where "…" denotes still higher orders, and an energy function can be written as follows:

$$E = \cdots -\frac{1}{3}\sum_i \sum_j \sum_k J_{ijk}^{(3)} S_i S_j S_k - \frac{1}{2}\sum_i \sum_j J_{ij}^{(2)} S_i S_j - \sum_i J_i^{(1)} S_i \tag{3}$$

provided that $J_{ijk}^{(3)} = J_{[ijk]}^{(3)}$ for *i*, *j*, *k* distinct, with $[\cdots]$ denoting permutations in cyclic order, and $J_{ijk}^{(3)} = 0$ for any *i*, *j*, *k* equal, and that similar symmetry requirements are satisfied for higher order connections. The updating rule maintains

$$S_i(t+1) = \mathrm{sgn}[h_i(t)]. \tag{4}$$

In the simple propositional case, logic clauses take the form $A_1, A_2, \ldots, A_n \leftarrow B_1, B_2, \ldots, B_m$ which says that $(A_1 \text{ or } A_2 \text{ or } \cdots \text{ or } A_n)$ if $(B_1 \text{ and } B_2 \text{ and } \cdots \text{ and } B_n)$; they are program clauses if $n = 1$ and $m \geq 0$: we can have rules, e.g., $A \leftarrow B, C$ saying $A \vee \neg(B \wedge C) \equiv A \vee \overline{B} \vee \overline{C}$, and assertions, e.g., $D \leftarrow$ saying that *D* is true.

A logic program consists of a set of program clauses and is activated by an initial goal statement. In Conjunctive Normal Form (CNF), the clauses contain one positive literal. Basically, logic programming in Hopfield model [7] can be treated as a problem in combinatorial optimization. Therefore, it can be carried out in a neural network to obtain the desired solution. Our objective is to find a set of interpretation (i.e., truth values for the atoms in the clauses which satisfy the clauses (which yields all the clauses true). In other words, we want to find 'models'.

The following algorithm shows how a logic program can be done in a Hopfield network based on Wan Abdullah's method:

(i) Given a logic program, translate all the clauses in the logic program into basic Boolean algebraic form.

(ii) Identify a neuron to each ground neuron.

(iii) Initialize all connections strengths to zero.

(iv) Derive a cost function that is associated with the negation of all the clauses such that $\frac{1}{2}(1 + S_x)$ represents the logical value of a neuron *X*, where S_x is the neuron corresponding to *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 *X* is false. Negation (neuron *X* does not occur) is represented by $\frac{1}{2}(1 - S_x)$; a conjunction logical connective is represented by multiplication whereas a disjunction connective is represented by addition.

(v) Obtain the values of connection strengths by comparing the cost function with the energy, *H*.

(vi) Let the neural networks evolve until minimum energy be reached. Then checked whether the solution obtained is a global solution.

The applied methodology may be summarized in the following way: given an optimization problem, find the cost function that describes it, design a Hopfield network whose energy function must reach (one of) its minima at the same point in configuration space as the cost function, so that the stable configurations of the network correspond to solutions of the problem. We do not provide a detail review regarding neural network logic programming in this paper, but instead refer the interested reader to Wan Abdullah [8]. Part of this section had been published in earlier works [9-12].

III. Normal Forms Representation

In Boolean logic, a formula is in conjunctive normal form if it is a conjunction of clauses, where a clause is a disjunction of literals, where a literal and its compliment cannot appear in the same clauses. As a normal form, it is useful in automated theorem proving. It is similar to the canonical product of sums form used in circuit theory.

(i) Disjunctive normal form (DNF)

A formula F is a Disjunctive Normal Form (DNF) if and only if F is of the form: $F = F_1 \vee F_2 \vee \cdots \vee F_n,\ n \geq 1,$ where each F_i is a conjunction of literal(s).

$F_1, F_2, ..., F_n,\ n \geq 1$ is its disjuncts.

(ii) Conjunctive normal form (CNF)

A formula F is a Conjunctive Normal Form (CNF) if and only if F is of the form:

$$F = F_1 \wedge F_2 \wedge \cdots F_n, \quad n \geq 1,$$

where each F_i is a disjunction of literal(s).

$F_1, F_2, ..., F_n,\ n \geq 1$ is its conjuncts.

In Boolean logic, CNF is much more commonly used in the area of logical reasoning systems. CNF is a method which is widely been used for standardizing and normalizing logical formulas. The main advantage of it is its uniformly formed form, which makes it suitable to automatic processing which needs to define the rule for machine learning to recognize the logic it operates.

Every propositional formula can be converted into an equivalent formula that is in CNF. This transformation is based on rules about logical equivalences: the double negative law, De Morgan's laws and the distributive law. Since all logical formulae can be converted into an equivalent formula in conjunctive normal form, proofs are often based on the assumption that all formulae are CNF. However, in some cases, this conversion to CNF can lead to an exponential explosion [13] of the formula. But, in our logic programming, we just handle with Horn clauses. So, satisfiability problem and explosion problem does not occurred [14].

We focused on five types of logical operations: negation, conjunction, disjunction, implication and equality.

- Logical negation $(\neg P)$ produces a value of true if its operand is false and vice versa. Logical conjunction $(P \wedge Q)$ produces a value of true if and only if both of its operands are true. Otherwise, they are all evaluated to false.
- Logical disjunction $(P \vee Q)$ produces a value of true if and only if at least one of its operands it true. Otherwise, it results to false.
- Logical implication $(P \rightarrow Q)$ produces a value of false just in the singular case the first operand is true and the second operand is false. The rest will give us a true evaluation.
- Logical equality $(P \equiv Q,\ P \leftrightarrow Q)$ produces a value of true if and only if both operands are false or both operands are true. Otherwise, it results to false.

$P \rightarrow Q$ is logically equivalent to $\neg P \vee Q$ and $P \leftrightarrow Q$ is logically equivalent to $(P \rightarrow Q) \wedge (Q \rightarrow P)$.

Table 1. Truth table for the basic Boolean operators

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

We can transform any logical formula into a normal form by applying the following rules:

(i) Use the laws:

$$F \leftrightarrow G = (F \rightarrow G) \wedge (G \rightarrow F)$$

$$F \rightarrow G = \neg F \vee G$$

to eliminate → and ↔

(ii) Repeatedly use the law:

$$\neg(\neg F) = F$$

and the De Morgan's laws:

$$\neg(F \vee G) = \neg F \wedge \neg G$$

$$\neg(F \wedge G) = \neg F \vee \neg G$$

to bring negation signs immediately before atoms.

(iii) Repeatedly use the distributive laws:

$$F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$$

$$F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$$

and the other laws as necessary.

So, by using De Morgan's laws, we can convert DNF clauses to CNF clauses. Let us look at an example for DNF to CNF conversion by using this law:

$$(A \wedge B) \vee (C \wedge D)(DNFFORM)$$

$$\equiv (A \vee (C \wedge D)) \wedge (B \vee (C \wedge D))$$

$$\equiv (A \vee C) \wedge (A \vee D) \wedge (B \vee C) \wedge (B \vee D)(CNFFORM).$$

IV. Satisfiability Aspects of Horn Clauses in CNF Representation

Satisfiability or SAT is a very basic problem in computer science. One way to solve SAT would be to try out every possible truth assignment. There are 2^n such assignments and l literals to set for each assignment. Such an approach requires $O(l \cdot 2^n)$ operations. So, in general, SAT is an NP-complete problem [15].

Deciding whether a given logical formula F is satisfiable constitutes the *satisfiability problem*. Being one of the hard computational problems, the satisfiability problem is very important both theoretically and practically. It plays a central role in the complexity theory as the seed of the class of NP-complete problems [16], and deciding satisfiability presents an inevitable and most frequent employed process in logic program.

Propositional satisfiability was the first problem shown to be NP-complete [17]. For Horn formula in CNF form, there is a more efficient algorithm to test satisfiability of a formula F. F is an example of Horn clauses in the CNF form

$$F = (\overline{X} \vee Y \vee \overline{Z}) \wedge (\overline{X} \vee Y) \wedge (X) \wedge (\overline{X} \vee \overline{Y} \vee Z)$$

$$\wedge (\overline{Z} \vee \overline{W} \vee \overline{X}) \wedge (W \vee \overline{Z} \vee \overline{Y}) \wedge (\overline{W} \vee \overline{U}).$$

Assume that X is true. We now see that certain clauses are satisfiable only if their respective positive literal is also made true. We can rewrite formula F in the context of the logic program as:

$$(X, Z \rightarrow Y) \wedge (X \rightarrow Y) \wedge (\rightarrow X) \wedge (XY \rightarrow Z) \wedge (ZWX \rightarrow) \wedge (ZY \rightarrow W) \wedge (WU \rightarrow).$$

For instance, with X being true $(\overline{X} \vee Y)$ is only satisfied if Y is made true. After setting X and Y to true, we notice that Z also needs to make true to satisfy $(\overline{X} \vee \overline{Y} \vee \overline{Z})$. We also need to set W to true to satisfy $(W \vee \overline{Z} \vee \overline{Y})$. Note that this process guarantees that all clauses containing at most one positive literal are satisfied by a minimal truth assignment. This implies that Horn clauses in CNF representation always satisfiable and solutions are guaranteed. This form also does not yield exponential explosion since the solutions searching tasks can be done in linear time. So, in our work, we can use Horn clauses in CNF to represent the logical clauses in doing logic programming without creating any satisfiability problem.

V. Comparison between CNF and DNF

In this section, we will use truth table analysis to compare the performance between CNF and DNF representation for the same set of logical clauses. Table 2, Table 3 and Table 4 illustrate the clauses with two literals, three literals and four literals, respectively.

Logical Clause: $Y \leftarrow X$

Truth Table 2(a). $\neg X \vee Y$ (DNF)

X	Y	$\neg X \vee Y$
1	1	1
1	–1	–1
–1	1	1
–1	–1	1

Truth Table 2(b). $\neg(X \wedge \neg Y)$ (CNF)

X	Y	$\neg(X \wedge \neg Y)$
1	1	1
1	–1	–1
–1	1	1
–1	–1	1

Logical Clause: $X, Z \leftarrow Y$

Truth Table 3(a). $(X \vee Z \vee \neg Y)$ (DNF)

X	Y	Z	$(X \vee Z \vee \neg Y)$
1	1	1	1
1	1	–1	1
1	–1	1	1
1	–1	–1	1
–1	1	1	1
–1	1	–1	–1
–1	–1	1	1
–1	–1	–1	1

Truth Table 3(b). $\neg(\neg X \wedge Y \wedge \neg Z)$ (CNF)

X	Y	Z	$\neg(\neg X \wedge Y \wedge \neg Z)$
1	1	1	1
1	1	–1	1
1	–1	1	1
1	–1	–1	1
–1	1	1	1
–1	1	–1	–1
–1	–1	1	1
–1	–1	–1	1

Clause: $X, Z \leftarrow Y, J$

Truth Table 4(a). $X \vee \neg Y \vee Z \vee \neg J$ (DNF)

X	Y	Z	J	$X \vee \neg Y \vee Z \vee \neg J$
1	1	1	1	1
1	1	1	–1	1
1	1	–1	1	1
1	1	–1	–1	1
1	–1	1	1	1
1	–1	1	–1	1
1	–1	–1	1	1
1	–1	–1	–1	1
–1	1	1	1	1
–1	1	1	–1	1
–1	1	–1	1	–1
–1	1	–1	–1	1
–1	–1	1	1	1
–1	–1	1	–1	1
–1	–1	–1	1	1
–1	–1	–1	–1	1

Truth Table 4(b). $\neg(\neg X \wedge Y \wedge \neg Z \wedge J)$ (CNF)

X	Y	Z	J	$\neg(\neg X \wedge Y \wedge \neg Z \wedge J)$
1	1	1	1	1
1	1	1	–1	1
1	1	–1	1	1
1	1	–1	–1	1
1	–1	1	1	1
1	–1	1	–1	1
1	–1	–1	1	1
1	–1	–1	–1	1
–1	1	1	1	1
–1	1	1	–1	1
–1	1	–1	1	–1
–1	1	–1	–1	1
–1	–1	1	1	1
–1	–1	1	–1	1
–1	–1	–1	1	1
–1	–1	–1	–1	1

From the tables, we can observe that the way to express the logical clauses in DNF is rather difficult compare to CNF. The number of evaluations needed for CNF is more than DNF. For an example, in Table 4, 48 evaluations are needed for DNF representation. On the other hand, for CNF, we just need 37 evaluations which are 20% less than DNF. When the number of literals per clause increased, we can observe that the computation time and steps get larger or more complex. So, more computation time and effort is needed for DNF evaluations. If we represent the clauses in DNF for doing logic programming in Hopfield network, then the chance for the neurons to get trapped in local minima is higher. Furthermore, the energy relaxation loop will also get complex. This will increase the processing or the running time. So, from the truth tables analysis and theory related to CNF representation, we can conclude that CNF presentation is better and more effective than DNF in doing logic programming in Hopfield network and also other types of knowledge or logical reasoning.

So, we convert logical clauses in DNF representation to CNF representation before doing logic programming in Hopfield network. We use Microsoft C++ platform to simulate the program.

VI. Simulation Result

Firstly, we generate random program clauses. We convert any clauses in DNF to CNF by using rules describe in Section III. Then, we initialize initial states for the neurons in the clauses. Next, we let the network evolves until minimum energy is reached. Then we test the final state obtained whether it is a stable state. If the states remain unchanged for five runs, then we consider it as stable state. Following this, we calculate corresponding final energy for the stable state. If the difference between the final energy and the global minimum energy is within tolerance value, then we consider the solution as global solution. Then we calculate hamming distance between stable state and global solution and ratios of global solutions.

We run the relaxation for 1000 trials and 100 combinations of neurons so as to reduce statistical error. The selected tolerance value is 0.001. All these values are obtained by try and error technique, where we tried several values as tolerance values, and selected the value which gives better performance than other values. Figures 1 to 6 illustrate the graphs obtained for ratio of global solutions and final hamming distances.

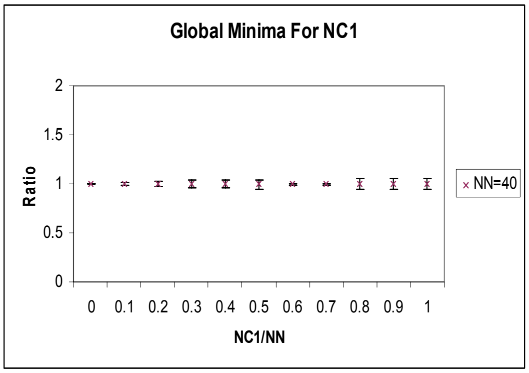

Figure 1. Global minima ratio for NC1.

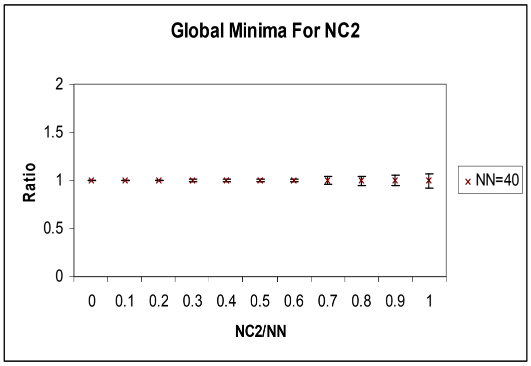

Figure 2. Global minima ratio for NC2.

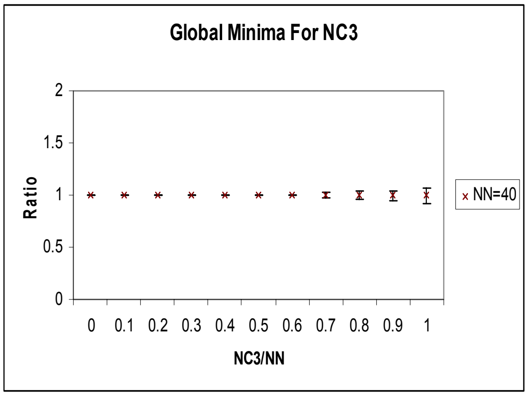

Figure 3. Global minima ratio for NC3.

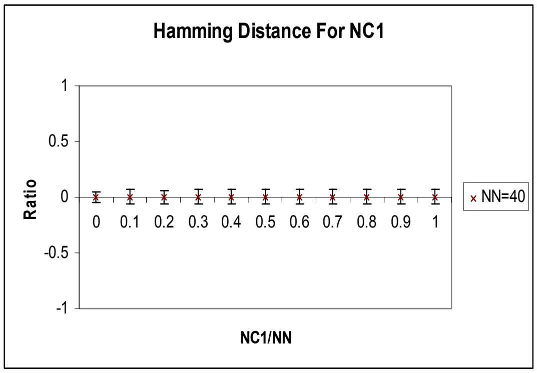

Figure 4. Hamming distance for NC1.

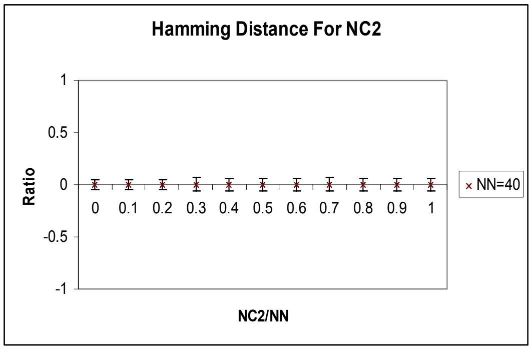

Figure 5. Hamming distance for NC2.

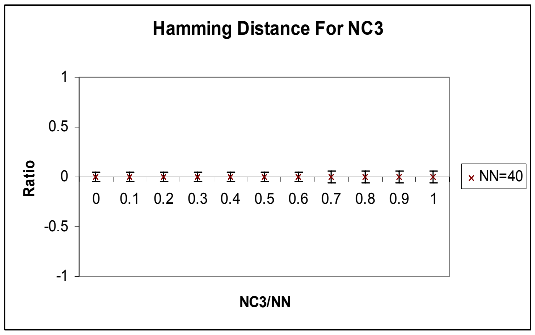

Figure 6. Hamming distance for NC3.

We can observe than the ration of global solutions is almost zero for all the trials. Since the graphs overlapped, so we just plot the results for $NN = 40.$ Most of the neurons which are not involved in the clauses generated will be in the global states. The random generated program clauses relaxed to the final states, which seem also to be stable states, in less than five runs. Furthermore, the network never gets stuck in any suboptimal solutions. This indicates good solutions (global states) can be found in linear time or less with less complexity.

Since all the solutions we obtained are global solution, so the distance between the stable states and the attractors are almost zero as shown in the figures. Supporting this, we obtained almost zero values for Hamming distance. So, the distance between the stable states and global states is almost zero. Since the loop of energy relaxation in doing logic programming consists only clauses in CNF form, so the network relaxed to global solutions without any problem. Although we increased the number of literals per clause but the complexity of the network never increased drastically due to we are just handling clauses in CNF. The neurons are able to jump the energy barrier to relax into global solutions. This indicates the energy landscape is nearly flat, which increased the capacity of the neurons to relax to global solutions. The ratio of global minima which is consistently around 1 shows that the CNF representation for the clauses are stable although the number of literals increased simultaneously. So, we can obtain models for the corresponding logical clauses without any computational complexity.

By carrying out computer simulations of the proposed method, we verified that by using CNF representation in doing logic program in Hopfield network, we can easily obtained models for the corresponding logic program.

VII. Conclusion

We had analyzed the performance of both CNF and DNF representation theoretically and analytically. By converting DNF into CNF, we found that the evaluation process can greatly be reduced as the time needed for evaluation is reduced. The advantage of the CNF shown here is that the time to process, computational complexity and number of runs for CNF clauses is much less than to process DNF in programming logic. Besides that CNF representation also does not yield any satisfiability problem for Horn clauses. As a conclusion, CNF representation is better than DNF representation in doing logic programming and knowledge reasoning in Hopfield network.

Acknowledgement

This research is partly financed by FRGS grant (203/PMATHS/671185) from the Ministry of Higher Education, Malaysia and RU grant (1001/PMATHS 817035) from Universiti Sains Malaysia.

References

[1] A. S. Avila Garcez, K. Broda and D. M. Gabbay, Neural-symbolic learning systems: foundations and applications, Perspectives in Neural Computing, Springer, 2002.

[2] J. J. Hopfield, Neural networks and physical systems with emergent collective computational abilities, Proc. Natl. Acad. Sci. USA 79(8) (1982), 2554-2558.

[3] Nikola Kasabov, Adaptation and interaction in dynamical systems; modeling and rule discovery through evolving connectionist systems, Appl. Soft Comput. 6 (2006), 307-322.

[4] Eyal Kolman and Michael Margaliot, Extracting symbolic knowledge from recurrent neural networks - A fuzzy logic approach, Fuzzy Sets and Systems 160 (2009), 145-161.

[5] S. Haykin, Neural Network: A Comprehensive Foundation, Macmillan, New York, 1998.

[6] W. A. T. Wan Abdullah, Neural network logic, O. Benhar et al., eds., Neural Networks: From Biology to High Energy Physics, Pisa: ETS Edit Rice, 1991, pp. 135-142.

[7] Saratha Sathasivam, Clauses Representation Comparison in Neuro-symbolic Integration, World Congress of Engineering, London, UK, 2008, pp. 34-37.

[8] W. A. T. Wan Abdullah, Logic programming on a neural network, Int. J. Intelligent Sys. 7 (1992), 513-519.

[9] Saratha Sathasivam, Logical content in the recurrent Hopfield network without higher order connections, European J. Scientific Research 37(3) (2009), 361-367.

[10] Saratha Sathasivam, Learning rule performance comparison in Hopfield network, American J. Scientific Research 6 (2009), 15-22.

[11] Saratha Sathasivam and W. A. T. Wan Abdullah, The satisfiability aspect of logic on Little Hopfield network, American J. Scientific Research 7 (2010), 90-105.

[12] Saratha Sathasivam, Upgrading logic programming in Hopfield network, Sains Malaysiana 39(1) (2010), 115-118.

[13] P. Miltersen, J. Radhakrishnan and I. Wegener, On converting CNF to DNF, Mathematical Foundations of Computer Science, 28th International Symposium, 2003, pp. 612-621.

[14] S. Sathasivam, Logic mining in neural network, Ph.D. Thesis, Malaysia, 2007.

[15] J. E. Hopcroft and J. D. Ullmann, Introduction to Automata Theory, Language and Computation, Addison-Wesley Publications, 1979.

[16] K. Iwama, CNF satisfiability test by counting and polynomial average time, SIAM J. Comput. 18 (1989), 385-391.

[17] S. Porschen, B. Randerath and E. Speckenmeyer, Exact 3-satisfiability in decidable in time $o(2^{0.16254n})$, Ann. Math. Artif. Intell. 43 (2005), 173-193.