Clauses Representation Comparison in Neuro-Symbolic Integration

Saratha Sathasivam

Abstract: **In the area of logical reasoning systems, conjunctive normal form (CNF) is used widely. Hence, we are interested in CNF form to ease the use in logical reasoning system. In this paper we show that given any clauses in DNF, we can convert it into respective CNF. We also prove the existence of logical equivalent between the DNF and the respective CNF conversion.**

Keywords**: logical reasoning, CNF, DNF**

I. INTRODUCTION

An artificial neural network (ANN) is an information processing paradigm that is inspired by the way the brain processes information at the low biological level. The collective behavior of a neural network then demonstrates the high-level behavior like ability to learn, recall, and generalize from training patterns or data [5]. Neural networks are becoming very popular with data mining practitioners, particularly in medical research, finance and marketing fields. This is because they have proven through comparison, their predictive power with statistical techniques using real data sets such as clustering technique, K-means algorithm and others. There is a lot of research in data mining based on neuro-symbolic integration (e.g. [2-4], see also [1]).

Wan Abdullah [9] 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 be either represented in Conjunctive Normal Form (CNF) or Disjunctive Normal Form (DNF). However, CNF is widely been used to represents clauses. In this paper, we prove the equalities between the CNF and DNF conversion in carrying out logic program in Hopfield network.

This paper is organized as follows. In section 2, the outline of doing logic programming on a Hopfield network is presented. Meanwhile, section 3 contains discussion regarding the CNF and DNF. Next, in section 4 contains simulation result using CNF and DNF conversion. Finally concluding remarks regarding this work occupy the last section.

Manuscript received March 5, 2010. 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.

Saratha Sathasivam. Author is with the Universiti Sains Malaysia (phone: 6046532428; fax: 6046570910; e-mail: saratha@cs.usm.my).

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 [5] $S_i(t),(i=1,2,....N)$. Neurons then 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,

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

which monotone decreases with the dynamics.

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

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

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

$$E = ... - \frac{1}{3}\sum_i \sum_j \sum_k J_{ijk}^{(3)} S_i S_j S_k -$$

$$\frac{1}{2}\sum_i \sum_j J_{ij}^{(2)} S_i S_j - \sum_i J_i^{(1)} S_i \qquad (3)$$

provided that $J_{ijk}^{(3)} = J_{[ijk]}^{(3)}$ for i, j, k distinct, with […] 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

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

In the simple propositional case, logic clauses take the form $A_1, A_2, ..., A_n \leftarrow B_1, B_2, ..., B_m$. which says that (A_1 or A_2 or …. or A_n) if (B_1 and B_2 and … 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 $A \vee \neg(B \wedge C) \equiv A \vee \bar{B} \vee \bar{C}$, 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.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Basically, logic programming in Hopfield model [9] 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 is reached. 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].

. III. INTRODUCTION OF NORMAL FORMS

A normal form for an expression is usually a subset of the standard syntax of expressions, such that either every expression can be rewritten in the normal form, or that expressions in the normal form have certain interesting properties. By restricting the form, we can often find simple and/or efficient algorithms for manipulating the expressions. There are two types of normal forms we are interested here: Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF). DNF is very commonly used in circuit design while CNF is much more commonly used in the area of logical reasoning systems [8].

In disjunctive normal form (or DNF), every expression is a disjunction of conjunctions of literals. A literal is a Boolean variable or its negation. In conjunctive normal form (or CNF), every expression is a conjunction of disjunctions of literals. A disjunction of literals is called a clause. In Boolean logic, the number of models for CNF is higher and reliable than the DNF conversion.

CNF is a method of 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 the machine to recognize the logic.

Note that all logical formulae can be converted into conjunctive normal form through repeated application of the distributive law of disjunction over conjunction, thus when making proves on formulae or on the structure of formulae, it is often convenient to assume that everything is in CNF.

(i) Disjunctive Normal Form (DNF)

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

$F_1, F_2, \ldots, F_n, n \geq 1$ 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:

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

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

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

Boolean expression is a statement that is either true or false. It is used to represent both DNF and CNF. Boolean data type has two values, *T* and *F* or '1' and '0'. Truth table [6] is a table showing all possible truth or false values for an expression representing a function as it lists all the possible combinations of inputs and outputs of the normal forms taken in consideration. It describes the relationship between the input and output of a Boolean function. Since each variable can take only two values, a statement with "*n*" variables requires a table with 2^n rows.

A. Truth table anaysis

After a long introduction of basic of normal forms and the conversion rules between DNF and CNF, now we are going to proof that DNF is not as good as CNF.

Table 1: CNF and DNF conversion

A	B	C	$\neg A$	$\neg C$	$\neg A \wedge B \wedge \neg C$	$\neg(\neg A \wedge B \wedge \neg C)$
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 Table 1, we see that the number of unsatisfied events for DNF conversion is higher than the CNF presentation for the same logical clauses evaluation. So,

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Table 2: True table for $A \vee \neg B \vee C \vee \neg D$

A	B	C	D	$\neg B$	$\neg D$	$A \vee \neg B \vee C \vee \neg D$
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	-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**

Table 3: True table for $\neg(\neg A \wedge B \wedge \neg C \wedge D)$

A	B	C	D	$\neg A$	$\neg C$	$\neg A \wedge B \wedge \neg C \wedge D$	$\neg(\neg A \wedge B \wedge \neg C \wedge D)$
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	1	-	-	**-1**	**1**
-1	-1	1	-1	-	-	**-1**	**1**
-1	-1	-1	1	-	-	**-1**	**1**
-1	-1	-1	-1	-	-	**-1**	**1**

From table 2 and 3 we compared number of unsatisfied events for CNF and DNF conversion. When the number of literals per clause increased, we observed that the number of unsatisfied events also increased. However CNF still can be consider stable. So, this indicates that CNF presentation is better than DNF for finding models for the corresponding logic program.

So, in our next work, we convert the DNF representation to CNF form before doing logic programming in Hopfield network. In next section, we look at the simulation result.

IV. SIMULATION RESULT

We focused on calculating the global minimum ratio (zM) and global Hamming distance (HDGlobal) of first, second and third order of a Boolean expression. First, user entered the amount of first, second and third order clauses that required. As we know, first order clause consists of one literal (neuron), second order clause consists of two literals and third consists of three neurons. For the clauses in DNF form, we convert it into CNF form before entering the task of doing logic programming in Hopfield network. Then, the converted clauses are put into the energy relaxation loop. Final states of the relaxed neurons resemble the corresponding model for the logic program.

The results we expect from running the program are global minimum ratio which is approximately 1 and Hamming distance which is approximately 0. The maximum number of neurons defined here is up to 40. It can be altered according to user's need by changing the initially defined NN value in the source code.

ISBN: 978-988-17012-9-9
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Table 4: Table of zM and HDGlobal with NN from 10 until 50

Number of neurons (NN)	Number of literals per clause (Nc1)	Number of literals per clause (Nc2)	Number of literals per clause (Nc3)	Global Minimum Ratio (zM)	Global hamming distance (HDGlobal)
10	5	5	5	0.99760002	0.00049830
15	8	8	8	0.99750000	0.00074675
20	10	10	10	0.99750000	0.00100466
25	13	13	13	0.99699988	0.00124954
30	15	15	15	0.99820000	0.00150402
35	17	17	17	0.99970001	0.00175012
40	20	20	20	0.99949998	0.00200351
45	22	22	22	0.99469100	0.00224691
50	25	25	25	0.99978003	0.00250521

From Table 4, we increased NN from 10 until 50 with interval 5. Then, Nc1, Nc2 and Nc3 are nearly to NN divided by 2. We noticed that when NN increased, zM is fluctuating; however, HDGlobal is slightly increased. When number of neuron and number of clauses increased, the time consumed is also increased when using Microsoft Visual C++ 6.0 and Dev-C++. The maximum number of neuron that we can run is only up to 50. This shows that time complexity getting larger and larger significantly. But when running using Linux, the maximum number of neuron that can be achieved is up to 90.

V. CONCLUSION

We have proved that DNF is not a good representation for logic program due to it's highly time consuming evaluation process. By converting it into CNF, we found that the evaluation process can be greatly reduced as the time needed for evaluation is reduced. The advantage of the CNF shown here is that the time to process CNF is much less than to process DNF in programming logic. In the logic programming, we added on a user-input function into the program, which will then base on the input to calculate the global minimum ratio and hamming distance. The result from the calculation will give us a global minimum ratio of approximately 1 and hamming distance which approximates to 0.

ACKNOWLEDGEMENT

Saratha Sathasivam thanks Universiti Sains Malaysia and Ministry of Higher Education.

REFERENCES

[1] A.S. Avila Garcez,, K. Broda, & D.M. Gabbay. Neural-Symbolic Learning Systems: Foundations and Applications. In *Perspectives in Neural Computing*, Springer, 2002.

[2] C.J. Mantas, J.M. Puche & J.M. Mantas . Extraction of similarity based fuzzy rules from artificial neural networks. *International Journal of Approximate Reasoning*, **43**, 2008, pp 202-221.

[3] Kasabov, Nikola. Adaptation and interaction in dynamical systems; Modelling and rule discovery through evolving connectionist systems. *Applied Soft Computing*, **6,** 2006**,** pp 307-322.

[4] Kolman Ehyal and Margaliot Michael.. *Extracting symbolic knowledge from recurrent neural networks- A fuzzy logic approach*. In press, 2008

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

[6] S. Sathasivam, *Logic Mining in Neural Network*. PhD Thesis, 2007, Malaysia.

[7] W.A.T.Wan Abdullah & S. Sathasivam,. Logic Mining Using Neural Networks. In *Proceedings of the International Conference in Intelligent System 2005 (ICIS 2005)*, 2005, Kuala Lumpur, Malaysia.

[8] W.A.T. Wan Abdullah. Logic Programming on a Neural Network. *Int .J. Intelligent Sys*, **7**, 1992, pp 513-519.

[9] W.A.T. Wan Abdullah. Neural Network logic. In O. Benhar et al. (Eds.), *Neural Networks: From Biology to High Energy Physics*. Pisa: ETS Editrice., 1991, pp. 135-142.