Malaysian Journal of Computer Science, Vol. 9 No. 1, June 1996, pp. 1-5

LOGIC PROGRAMMING IN NEURAL NETWORKS

Wan Ahmad Tajuddin bin Wan Abdullah
Jabatan Fizik
Universiti Malaya
50603 Kuala Lumpur
Malaysia
Tel.: 603-7594192
Fax: 603-7594146
email: wat@cc.um.edu.my

ABSTRACT

Logic programming is carried out on a neural network. A higher-order Hopfield neural network is used to minimise logical inconsistency in interpretations of logic clauses or programs. The connection strengths are defined from the logic program; the network relaxes to neural states corresponding to a valid (or near-valid) interpretation. 'Creativity' can be thought of as the crossing of configurational energy barriers to arrive at alternative interpretations. The formalism allows the incorporation of non-monotonicity; non-integral degrees of truth in rules; and non-Horn clauses. Hebbian learning in an environment with some underlying logical rules governing events is equivalent to hardwiring the network with these rules.

Keywords: ***Hopfield neural network, Higher-order connections, Logic program-ming, Combinatorial optimisation, Hebbian learning***

1.0 INTRODUCTION

Neural networks or connectionist architectures provide an alternative computational paradigm [1], and can be seen as a step towards the understanding of intelligence. It departs from the traditional von Neumann 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 [2].

There are various versions of the neural network, of which the symmetric (of connections), densely-connected, Hopfield model [3] is one. This network has been shown to evolve in such a way as to minimise a configurational energy function and can thus be used for solving combinatorial optimisation problems [4]. Adequately good solutions can be found in linear time or less. Also, optimisation is the basis of many (if not all) intelligent processes.

It can be shown that prepositional logic programming can be done on the model of a single neuron [5]. Recently [6], we have also proposed a method of doing logic programming on a Hopfield neural network (with higher-order connections) through the minimisation of logical inconsistencies. Consequently, by looking at this process backwards, we showed that [7] synaptic changes from learning correspond to the formation of logical rules. This paper presents a review of these ideas, and we see how the formalism allows extensions to the Horn clause logic usually employed and discuss the ramifications of the learning of logical rules.

2.0 NEURAL NETWORKS

A neuron i can formally be modelled as a two-state element S_i (here we take them to be bipolar, *i.e.* $S_i \in \{-1,1\}$) whose state depends on the input from other neurons j via connections J_{ij} of various strengths (positive or negative):

$$S_i := \text{sign}(\Sigma_j J_{ij} S_j).$$

If J_{ij} is zero-diagonal and symmetric, we may write a configurational "energy" [3]

$$E = -1/2 \ \Sigma_i \Sigma_j J_{ij} S_i S_j$$

which is monotone decreasing with the evolution of S). Thus, depending on the initial neural configurations, the system evolves into a configuration for which E is a minimum.

By mapping neurons to 'switches' indicating choices in a combinatorial optimisation problem, we can arrive at the combination with least cost if we equate E to the cost function associated with the problem (plus perhaps functions corresponding to constraints on the choices) and thereby define the values for connection strengths, and allow the network to relax. The global minimum provides the solution to the problem. Difficult problems may be solved adequately well in a linear or shorter time in this way. The complexity which is spread over time in a sequential machine is spread over space in the massively-connected network.

In some cases, networks with multiconnections or higher-order interactions between neurons [8-14] are needed. This occurs when the cost function requires the expression for energy to have, for example, a third order term in neural values, $- 1/3 \Sigma_i \Sigma_j \Sigma_k J^{(3)}_{ijk} S_i S_j S_k$, modifying the dynamics to (if for completeness a first order term, $- \Sigma_i J^{(1)}_i S_i$, is included as well)

$$S_i := \text{sign}(\Sigma_i \Sigma_j J^{(3)}_{ijk} S_j S_k + \Sigma_i J^{(2)}_{ij} S_j + J^{(1)}_i)$$

provided that $J^{(3)}_{ijk}$ is zero whenever any of its indices are equal, and $J^{(3)}_{ijk} = J^{(3)}_{[ijk]}$, where $[ijk]$ denotes any permutation of i, j, k. (We have included superscripts to differentiate between connections of different orders). Multiconnections can actually be simulated by extra neurons: the product $S_j S_k$ is given by S_l if there is an exclusive-or network from S_j and S_k to S_l. Optimisation using multi-connected networks have been discussed [12, 15].

We show below how logic programming can be interpreted as a problem of optimisation and implemented on a neural network.

3.0 LOGIC PROGRAMMING

In logic programming [16], a set of Horn clauses (logic clauses of the form A C B_1, B_2, ..., B_N where the arrow may be read 'if' and the commas 'and') are given and the aim is to find interpretations (i.e. truth value assignments) for the atoms (A, B_1, etc.) in the clauses which are consistent with the clauses (which yields all the clauses true). In essence logic programming can be seen as a problem in combinatorial optimisation and it can be carried out on a neural network by the procedure above. This is done by using the neurons to store the truth values of the atoms and writing a cost function which is minimised when all the clauses are satisfied.

As a prepositional example, consider the logic program below:

$A \leftarrow B, C.$
$D \leftarrow B.$
$C \leftarrow.$

whose three clauses translates as $A \vee \neg (B \wedge C) = A \vee \neg B \vee \neg C$, $D \vee \neg B$, and C respectively. The clauses have implied conjunction between them,

$P = \quad A \leftarrow B, C$
$\wedge \quad D \leftarrow B$
$\wedge \quad C \leftarrow.$

so given a goal

$\leftarrow G.$

we require to show that $P \wedge \neg G$ is inconsistent in order to proof the goal. Alternatively, we require to find an interpretation for the Herbrand base of the problem which is consistent with P (i.e. which yields P *true*) and examine the truth of G in such an interpretation. If we assign the values 1 to *true* and 0 to *false* then $\neg P = 0$ indicates a consistent interpretation while $\neg P = 1$ reveals that at least one of the clauses in the program is not satisfied. Therefore, looking for a consistent interpretation is a combinatorial (of assigning truth values to ground atoms) minimisation of the 'inconsistency', the value of $\neg P$.

3.1 Logic Programming on Neural Networks

As was mentioned, logic programming can be seen as a problem in combinatorial optimisation and thus it can be carried out on a neural network. The neurons store the truth values of the atoms and we require to write a cost function which is minimised when all the clauses are satisfied. This function has to represent $\neg P$ so that minimisation of the cost function minimises the 'inconsistency' as discussed above. To obtain an arithmetic function, we represent logical 'and' as multiplication, and without loss of consistency since we are minimising the function and since the form of $\neg P$ is a disjunction of conjunctions, logical 'or' as addition.

Since in our example

$$\neg P = (\neg A \wedge B \wedge C) \vee (\neg D \wedge B) \vee (\neg C)$$

we write the cost function to be minimised as follows:

$$E_P = (1/8)(1-S_A)(1+S_B)(1+S_C) + (1/4)(1-S_D)(1+S_B) + (1/2)(1-S_C)$$

where the neurons S_A, etc. represent the truth values of A, etc., with $S_A = 1$ if A is true, and -1 otherwise. As we have chosen arithmetic addition to represent logical disjunction, the value of E_P depends on the number of clauses satisfied by the interpretation - the more the clauses unsatisfied, the bigger the value of E_P. Minimum E_P corresponds to the 'most consistent' selection of truth value assignments. The cost function, when programmed onto a third order neural network, yields

$J^{(3)}_{ijk}$ = 1/16 if I, j, k = A, B, C in any order
= 0 otherwise
$J^{(2)}_{ij}$ = -1/8 if I, j = B, C in any order
= 1/8 if I, j = A, B, or A, C, in any order
= 1/4 if I, j = D, B in any order
= 0 otherwise
$J^{(1)}_i$ = -1/4 if i = B
= 1/4 if i = D
= 1/2 if i = C
= 0 otherwise

Notice that with respect to the cost function, the addition of more rules or facts to the database is simply additive in $J^{(2)}_{ij}$, etc. Notice also that Multiconnections are needed when there are long clauses like the first one in the example logic program.

So far we have not mentioned variables. One way of dealing with universal quantification is to replace all clauses with variables with the corresponding clauses with all possible instantiations of the variables. The number of clauses would then multiply; however, on a neural

network we have the advantage of having as many neurons as we want without causing performance to deteriorate. Neurons can then represent whole (variable-free) relations like rel1(arg1,arg2), rel1(arg1,arg3), ..., rel2(arg1,arg2), etc.

The above example illustrates how we may do logic programming on a neural network. The following procedure is subscribed to:

(1) List all clauses. Clauses with variables are replaced by all possible instantiations from the Herbrand base.

(2) Identify a neuron to each ground atom.

(3) Initiate all connection strengths to zero. Multiconnections of up to degree p are needed where p is the length (no. of ground atoms) of the longest clause. For each clause, modify the appropriate connections in a similar way as demonstrated in the example above.

(4) Initiate neural states to probable values where known (e.g. when the truth of a ground atom is given by an assertion), or to random values otherwise.

(5) Let the neural network evolve until an energy minimum is reached. The neural states then provide a solution interpretation for the logic program, and the truth of a ground atom in this interpretation may be checked. If the goal sought involved variables, then the set of appropriate ground atoms are checked.

A thing to note is that the neural network would also yield local minima as well as the global minimum, depending on the initial configuration. One way to escape local minima in order to find the global minimum is to resort to simulated annealing [17, 18], where we include "thermal" noise to allow the system to climb over energy barriers around local minima. Other methods of seeking global optima have been compared elsewhere [19].

Depending on the initial configuration, the network may arrive at alternative solutions or at near-solutions (interpretations with logical inconsistency almost zero). We may thus understand 'creativity' as the ability or ease in escaping the energy minimum corresponding to stereotype solutions and arriving, perhaps many barriers away, to a minimum corresponding to a 'novel' solution.

This formalism for logic programming also raises some other interesting points which we discuss in the following section.

3.2 Generalisations to First-Order Horn Clause Logic

The formalism above attempts to find solutions with least inconsistency - it does not require 'correct' solutions. This allows solution in the presence of incomplete knowledge. The procedure hunts for the best solutions, given the clauses in the logic program, and these solutions may change as clauses are added (i.e. as connection strengths are modified). Even when clauses in the logic program are inconsistent with each other, the neural network still proposes a solution, which is an interpretation which gives the least logical inconsistency as defined by the energy function. It can therefore deal with non-monotonicity.

Because of the additive nature of the inconsistency cost function, we can introduce weightings to clauses. We have been taking a unit of logical inconsistency as equivalent to a clause being unsatisfied, but this can be altered and clauses be given different weightings in the energy function, by multiplying the appropriate terms by the respective weighting factors, to reflect their relative 'degrees of truth'. Clauses with larger weightings tend to be more often satisfied than clauses with smaller weightings. This has also something to do with learning as discussed in the next section.

Our formalism has not imposed any restrictions on the type of clauses acceptable in the logic program. Thus non-Horn clauses are also allowed; they only need to be encoded properly in the energy function. The energy function should eventually have the form of a sum of terms each of which corresponds to a clause, and each of which is 0 when the clause is satisfied and 1 otherwise: the final form of the clauses should thus be disjunction (so that their negation in the inconsistency function would yield conjunctions) - thus some clauses may be broken up into several clauses in the final form.

Also, by dealing with universal quantifications in the manner we have done, higher-order logics may also be incorporated by treatment in the same manner. Variables representing predicates, for example, can be allowed. These variables are then instantiated over all possible values in the same manner as for those representing arguments in predicates.

Of course, with the neural network formalism, one interesting aspect to investigate is that of learning. Below, we discuss implications for logic programming.

4.0 LEARNING

Learning in a neural network corresponds to the modification of connection strengths which would yield a modified behaviour supposedly better (in the sense of correctness of response, etc.) than previously. One principle of learning which has been suggested is that of Hebbian learning [20] where the strength of a connection is increased when it is in frequent usage, i.e.

$$\Delta J^{(2)}_{ij} = a_2 S_i S_j$$

for a two-neuron synaptic connection, where a_2 is a constant corresponding to the learning rate. From this basis, associative memory can be obtained [3]. We may generalise this to interactions of any order by

$$\Delta J^{(n)}_{ij..n} = a_n S_i S_j ... S_n$$

where a_n is the respective learning rate.

We showed that [7] synaptic changes from learning correspond to the formation of logical rules. For our example above, assume that events corresponding to A, B, C and D occur uniformly randomly, with the provision that the clauses in the program are satisfied. This means that, considering the first clause, conjunctions of all truth value combinations of A, B and C occur except for the combination $<S_A = -1,\ S_B = 1,\ S_C = 1>$ which violates the clause and the equal frequencies of occurrences of the other possible combinations would render, by the expression above, a relative increase in $J^{(3)}_{ABC}$ (and similarly in $J^{(3)}_{ACB}$, etc.) of $a_3 N_e/8$, where N_e is the number of occurring events. Looking at the sub-combinations B and C, A and B, and A and C, it can similarly be seen that the non-occurrence of the disallowed combination results in $J^{(2)}_{BC}$ (and $J^{(2)}_{CB}$) changing by $-a_2 N_e/8$, $J^{(2)}_{AB}$ (and $J^{(2)}_{BA}$) by $a_2 N_e/8$ and $J^{(2)}_{AC}$ (and $J^{(2)}_{CA}$) by $a_2 N_e/8$. (More details may be found elsewhere [21]). Similar considerations also show that $J^{(1)}_A$ increases by $a_1 N_e/8$ while $J^{(1)}_B$ and $J^{(1)}_C$ decrease by the same amount. Notice that from these changes, it may be interpreted that, by comparing with the direct assignments to the synaptic strengths above, the system has *learnt* the clause A ← B, C just from the non-occurrence of the disallowed combination, with the weightage N_e, provided that $a_n = 1/(n\text{-}1)!$ For the second and third clauses, similar analyses would show that the respective clauses are learnt in the same manner. Generalisation to other clauses and other orders is straightforward. Thus, Hebbian learning in environments obeying underlying logical rules as given by respective clauses is equivalent to hardwiring the network with these respective rules.

This also allows a way of dealing with non-monotonicity: using Hebbian learning, we can let the rules be formed (strengthened or weakened) on their own, from the activity of neurons representing the relevant ground atoms. This is alternative to "consciously" dealing with non-monotonicity by straightforward addition or deletion of clauses using the same procedure as for initiating the connection strengths given the set of clauses in a logic program: addition to or subtraction from, of appropriate positive values, the appropriate connection strengths.

5.0 CONCLUSION

In conclusion we have shown how logic programming may be carried out on an optimising neural network and how clauses or rules may be learnt by a neural network just from the observation of the occurrences of events obeying these rules. We have written and tested a neural network theorem prover (i.e. an interpreter of logic programs into neural network architectures, and the simulator for such neural network) in C, named CLOG [22], which runs on a personal computer. CLOG is of course a simulator which updates neurons serially, but optimums are arrived at in typically 2 updates/neuron for modest-sized logic programs. For the program tested, the global optimum was obtained in 11 out of 16 trials. Of date, rigorous testing has yet to be carried out.

We have also written and run a program which carries out Hebbian learning in environments with underlying rules. One interesting use of the results presented in this paper is the extraction of underlying logical rules in data through Hebbian learning on a neural network.

On a more esoteric note, the result concerning the learning of implicit rules gives a modern perspective to the associationist epistemology of Al-Ghazali and others, as discussed elsewhere [23, 24].

ACKNOWLEDGMENT

This research was supported in part by MPKSN grant R&D 4/41/01.

REFERENCES

[1] W. A. T. Wan Abdullah, "The Connectionist Paradigm", in *Proceedings 1st National Computer Science Conference, Kuala Lumpur, January 1989*, pp. 95-111.

[2] W. A. T. Wan Abdullah, "*Computations with Neural Networks and Neural-Network-like Structures*", in: J. Noye and C. Fletcher (eds.), "*Computational Techniques and Applications: CTAC-87*", Elsevier-North Holland, Amsterdam, 1988.

[3] J. J. Hopfield, "*Neural networks and physical systems with emergent collective computational abilities"*, in *Proceedings National Academy of Science USA*, Vol. 79, 1982, pp. 2554-2558.

[4] J. J. Hopfield and D. W. Tank, "'Neural' Computation of Decisions in Optimization Problems", *Biol. Cybern.*, Vol. 52, 1985, pp. 141-152.

[5] W. A. T. Wan Abdullah, "Biologic", *Cybernetica*, Vol. 31, 1988, pp. 245-251.

[6] W. A. T. Wan Abdullah, "Logic Programming on a Neural Network", *Int. J. Intelligent Syst.*, Vol. 7, 1992, p. 513.

[7] W. A. T. Wan Abdullah, "The Logic of Neural Networks", *Phys. Lett.*, Vol. 176A, 1993, p. 292.

[8] P. Baldi and S. S. Venkatesh, *Phys. Rev. Lett.*, Vol. 58, 1987, p. 913.

[9] E. Gardner, "Multiconnected neural network models", *J. Phys. A: Math. Gen.*, Vol. 20, 1987, p. 3453.

[10] L. F. Abbott and Y. Arian, *Phys. Rev. A*, Vol. 36, 1987, p. 5091.

[11] L. Personnaz, I. Guyon and G. Dreyfus, *Europhys. Lett.*, Vol. 4, 1987, p. 863.

[12] W. A. T. Wan Abdullah, "Dendritic trees and non-quadratic combinatorial optimisation*"*, *Malaysian J. Sci.*, Vol. 9, 1987, pp. 105-109.

[13] G. A. Kohring, *J Phys France*, Vol. 51, 1990, p. 145.

[14] R. M. C. Almeida and J. R. Iglesias, *Phys. Lett.*, Vol. 146A, 1990, p. 239.

[15] H. Mueller-Krumbhaar, *Europhys. Lett.*, Vol. 7, 1988, p. 479.

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

[17] S. Kirkpatrick, C.D. Gelatt and M. P. Vecchi, "Optimization by simulated annealing", *Science*, Vol. 220, 1983, pp. 671-680.

[18] S. Geman and D. Geman, "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images", *IEEE Transactions Pattern Analysis & Machine Intell.*, Vol. PAMI-6, 1986, pp. 721-741.

[19] W. A. T. Wan Abdullah, "Seeking Global Minima", *J. Comput. Phys.*, Vol. 110, No. 2, 1994, p. 320.

[20] D. O. Hebb and D. O., "*The Organisation of Behaviour*", Wiley, New York, 1949.

[21] W. A. T. Wan Abdullah, "*Neural Network Logic*", in: O. Benhar, C. Bosio, P. del Giudice and E. Tabet (eds.), Neural Networks: From Biology to High Energy Physics, ETS Editrice, Pisa, 1991, pp. 135-142.

[22] W. A. T. Wan Abdullah, "Pengaturcaraan Logik Menggunakan Rangkaian Neuron", *Pros. Seminar Sains Komputer, Serdang, Jun 1994*, pp. 140-145.

[23] W. A. T. Wan Abdullah, "A Connectionist Epistemology", *Cybernetica*, Vol. 34, 1991, p. 75.

[24] W. A. T. Wan Abdullah, "Neural Networks, Logic Programming and Al-Ghazali's Epistemology", in *Proceedings AMR-IT 1991, Leicester, U. K., 1991.*

BIOGRAPHY

Wan Ahmad Tajuddin bin Wan Abdullah received his Ph.D in High Energy Physics from Imperial College, University of London, in 1985. He is currently an Associate Professor in the Department of Physics, Universiti Malaya, with interests in bioinformatics and bioorganisation. He is a member of Institut Fizik Malaysia, International Neural Network Society and Artificial Intelligence Society.