Proceedings of the 6th WSEAS Int. Conf. on NEURAL NETWORKS, Lisbon, Portugal, June 16-18, 2005 (pp272-277)

A fault tolerant Connectionist Model for Predicate Logic Reasoning, Variable Binding: Using Coarse-coded Distributed Representations

SRIRAM.G. SANJEEVI, Dr. PUSHPAK BHATTACHARYA

Sriram. G. Sanjeevi, Asst.Professor,
Dept. of Comp. Science & Engg.,
N.I.T. Warangal, Warangal 506004
INDIA
91-0870-2430440

Dr. Pushpak Bhattacharya, Professor,
Dept. of Comp. Science & Engg.,
I.I.T. Bombay, Mumbai 400076
INDIA
91-22-5767718

Abstract: - In this paper, we describe a fault-tolerant model for reasoning using forward chaining for predicate logic rules and facts with *coarse-coded* distributed representations of instantiated predicates in a connectionist frame work. Distributed representations are known to give advantages of fault tolerance and graceful degradation of performance under noise conditions. The system supports usage of complex rules which involve multiple conjunctions and disjunctions. The system solves the variable binding problem in a novel way using coarse-coded distributed representations of instantiated predicates without the need to decode them into localist representations. System's performance with regard to its ability to exhibit fault tolerance under noise conditions is studied. The system offers better results of fault tolerance under noise conditions as compared to a connectionist reasoning system which uses localist representations. It has also exhibited better fault tolerance as compared to a Hopfield net for error correcting tasks of same magnitude.

Key-Words: - Coarse-coding, Predicate, Connectionist, Reasoning, Fault tolerance, Variable binding.

1 Introduction

Traditionally reasoning systems using predicate logic have been implemented using symbolic methods of artificial intelligence. Connectionist methods of implementation of reasoning systems describe an alternative paradigm. Among the connectionist systems they use two types of representational schemes. They are 1) localist and 2) distributed representational schemes.

Localist representational schemes represent each concept with an individual unit or neuron. In the distributed representational schemes each unit or neuron is used in representation of multiple concepts and multiple units or neurons are used to represent a single concept. In the literature, some localist methods for reasoning using connectionist networks have been described. The connectionist inference system *SHRUTI* [1], [2], [3] described a localist method where temporal synchrony was used to create bindings between variables and entities they represent. A variable x of the predicate *give(x, y, z)* is getting bound to an entity d if the nodes representing them fire during the same phase of time *p1* during the predicate p activation period T. The time period T is divided into three phases *p1, p2* and *p3* during which synchronous firing of variable x, y and z and entity nodes they bound respectively takes place. This method has used temporal synchrony as a mechanism to establish variable binding. *CONSYDERR* [4] described a localist method for variable binding and forward reasoning. It uses an assembly or a set of interconnected nodes to represent each predicate $p(x_1.....x_k)$. Each assembly contains one C node for storing the confidence value of the predicate p and k X nodes to store the binding values for k variables of the predicate p. A separate node is allocated for each variable of a predicate. Each such node stores a value representing a particular object being bound with that variable. Different objects being bound to a variable will be

given separate values. Since, these systems use localist representations, advantages of distributed representations [5] are not obtainable by them and hence the motivation for a distributed representation based reasoning system. It is investigated here in this work as to what advantages are obtained by a distributed representation based reasoning system over their localist counter parts. Further we deal with the issue of how variable binding may be accomplished in such a connectionist environment which uses distributed representations of its instantiated predicates.

2 Rule and Fact Base

Our system represents and reasons with predicate logic rules and facts. Following are rules and facts we use.

1. give(x, y, z) —> own(y, z);
2. buy(x,y) —> own(x,y);
3. own(y, z) —> donate(y, z);
4. own(y,z)∧ wantstobuy(w,z)∧
(hasrequiredmoney(w, m)∨
hasgoodcreditrating(w)) —> cansell(y,w,z);
5. give(John,Mary,Book-1);
6. give(John,Chris,Book-2);
7. wantstobuy(Walter,Book-2);
8. hasrequiredmoney(Walter,Money);
9. hasgoodcreditrating(Walter);

Our system uses the above rule base and makes inferences shown below.

1. own(Mary,Book-1);
2. donate(Mary,Book-1);
3. own(Chris,Book-2);
4. cansell(Chris,Walter,Book-2);

Our task is to start with the above knowledge base and obtain the results of inferencing correctly by our reasoning system. In this paper we see how to accomplish the forward reasoning for predicate calculus facts and rules using neural networks which operate on coarse coded distributed representations. We start with above database consisting of predicate logic facts and rules. Each fact of predicate p_i is represented by a vector $\mathbf{v_{ij}}$.The vector $\mathbf{v_{ij}}$ is a k dimensional vector which stores the coarse coded representation of a predicate fact. The different instantiations of predicate p_i are each represented by separate vector $\mathbf{v_{ij}}$ where j varies from *1* to m. Thus the set of instantiated predicates of p_i are represented by a subset of vector space R^k. The value of i depends on the number of predicates in the rule base. We then design suitable connectionist frame work for doing reasoning with forward chaining for the rule and fact base using these vectors.

3 Forward Reasoning using Connectionist System

We describe here how forward reasoning using localist representations [6], [7] are made using a connectionist system. Let us consider the *rule 1:give(x,y,z)—> own(y,z)* from the knowledge base. We define how localist representations be made for the values getting bound to the variables *x*, *y* and *z* and how the localist connectionist system makes inference from the rule. We assign values to each variable on the left hand side of a rule. A value is allocated to a specific variable and it will represent a particular object getting bound to that variable. We assign binary string which is the localist representation of object getting bound to that variable. Suppose, we have three different objects for possible binding to variable *x*. We encode them by the localist patterns *001, 010*, *100* respectively. Other variables *y* and *z* also get similar localist patterns for being assigned to them. We need a pattern code for distinguishing among predicates. We assign an *n* bit binary code to distinguish among *n* predicates. If *n = 4* say then our binary pattern will be *0001* to designate the predicate under consideration *give*. Then we choose pattern *0010* to represent predicate *own*. We also need a truth value allocated for a predicate. We assign a single bit which could be *1* or *0* denoting predicate fact being true or false respectively. Then the localist pattern for the LHS of our rule can be written as: *0001 001 001 001 1.*

The first 4 bit value denotes the predicate *give*, the next 3 bit value denotes an object getting bound to variable *x* and the next 3 bit value denotes an object getting bound to variable *y* and so on. The last bit indicates the truth value of predicate *give*.

We assigned values '001', '*001*' and '001' to variables *x, y* and *z* respectively. These values represent objects which are getting bound to these variables, say, *John, Mary* and *Book-17.* We have instantiated thereby the variables *x, y* and *z* of the LHS of the rule 1.

This will activate rule 1 and make variables on the right hand side of the rule *'y'* and *'z'* be assigned the values '*001*' and '*001*' representing the objects *Mary* and *Book-1* respectively. This asserts the right hand side of the rule 1, which is own*(Mary, Book-1).*

Because of the rule activation the localist pattern representation for RHS will be after the rule firing *0010 001 001 1*.

When a number of such rules are cascaded we continue the process of forward chaining to do the forward reasoning. When RHS of rule 1 is asserted it activates rule 3 and part of rule 4. Rule 3 gets activated and fires and *own* of rule 4 gets activated and provided other parts *wantstobuy* and *hasrequiredmoney* or *hasgoodcreditrating* of rule 4 also get activated then rule 4 fires.

3. own(y, z) —> donate(y, z);

4. own(y,z)∧ wantstobuy(w,z)∧

(hasrequiredmoney(w, m)∨

hasgoodcreditrating(w)) —> cansell(y,w,z);

The binding information is similarly passed on in these rules for the variables. This way forward reasoning is accomplished using localist representations. In *Table 1* and *2* below we show samples of localist vectors for some of the predicates in the rule base.

Table 1. Shows a sample of localist tuples used by predicate *give*

S.No of Tuple	215
Predicate 'id' code	00001000000
Localist Value of x	0000100000
Localist Value of y	0000100000
Localist Value of z	0000010000
Truth Value of Predicate	00001

Table 2. Shows a sample of localist tuples used by predicate Cansell

S.No of Tuple	46
Predicate 'id' code	00000001000
Localist Value of y	0000000010
Localist Value of w	0000100000
Localist Value of z	0000010000
Truth Value of Predicate	00001

3.1 Obtaining Coarse-coded Distributed Representations from Localist Representations

Consider the following tuple from the localist representation table of predicate *give(x, y, z)*,

'00001000000 0000000001 0000000010 0000000010 00001'.

We view the above vector as being kept in overlapping coarse zones of length of 4 consecutive bits and encode the zone as *1* if there is at least one *1* bit in that zone or else as *0*. We then consider next coarse zone and encode it as *1* or *0* following the above method. We do this process left to right starting from the left most bit. We do this encoding process for above localist tuple to get the following coarse-coded tuple

' 01111000000 0000001111 0000011110 0000011110 01111'.

Coarse-coding can be applied when the number of *1*'s in the original string is sufficiently sparse. If the number of *1*'s in the original string is not sufficiently sparse then coarse-coded string when decoded will not yield the original string. This is the reason we have chosen a 5 bit string to denote the truth value of predicate(in which first 4 bits were kept as zeros). The reason the coarse-coding could be applied successfully to our reasoning problem is that localist representations of instantiated predicates were sufficiently sparse with regard to distribution of *1*'s. Coarse-coding increases the information capacity [5] by increasing the number of units active at a time compared to localist codes which have sparsely populated *1*'s. The amount of information conveyed by a unit that has a probability *p* of being '*1'* is

$$-p\log(p) - (1-p)\log(1-p).$$

We obtain the coarse-coded representations of tuples for all the *predicates* in the *rule base* using the above described method. We show here a sample of coarse-coded representations of the tuples for predicates *give* and *cansell* in the rule base.

Table 3. Shows a sample of Coarse-code Representation of data tuples used by predicate *give*

S.No of Tuple	215
Predicate 'id' code	01111000000
Value of x	0111100000
Value of y	0111100000
Value of z	0011110000
Truth Value of Predicate	01111

Table 4. Shows a sample of Coarse-code Representation of Data Tuples used by predicate *cansell*

S.No of Tuple	46
Predicate 'id' code	00001111000
Value of x	0000011110
Value of y	0111100000
Value of z	0011110000
Truth Value of Predicate	01111

3.2 Organization of Neural Networks in the Connectionist Reasoning System

The neural networks accomplish the forward reasoning using the coarse-coded tuples. They generate inferences by firing rules from the rule base. Consider the neural networks shown in figure 1. When impressed on its inputs with one of the vectors $\mathbf{v_g}$ from the predicate table *give* the network 1 generates on its outputs a vector $\mathbf{v_o}$ from the predicate table *own*. This way the rule *give*(*x,y,z*) —> *own*(*y,z*) was processed. This in turn impresses on the inputs of network 2 to generate a vector $\mathbf{v_d}$ on its outputs. This processed the rule *own(y,z)* —> *donate (y,z).* These vectors are in coarse-coded form and denote a predicate fact. So we see the rules 1 and 2 are getting activated in a forward chaining fashion .

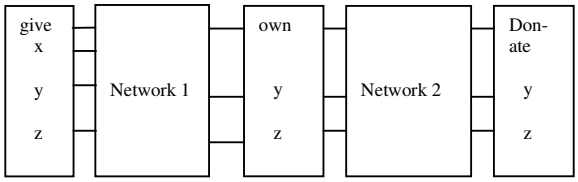

Fig. 1. Neural Networks for processing rules 1 and 2

Similarly impressing a vector $\mathbf{v_b}$ on the inputs of network 3 generates vector $\mathbf{v_o}$ on its outputs and this in turn gets impressed on inputs of network 4 generating on its outputs vector $\mathbf{v_d}$ from the predicate table *donate*. This way the rules *buy (y,z)* —> *own (y,z)* and *own (y,z)* —>*donate (y,z)* get processed in a forward chaining fashion generating new inferences . The networks 3 and 4 are shown in figure 2 accomplishing this activity.

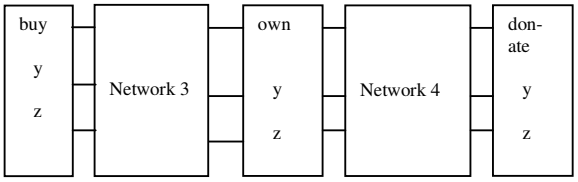

Fig. 2. Neural Networks for processing rules 3 and 4

3.3 Variable Binding during processing of the Complex Rule

Consider the following complex rule which involves multiple conjunctions and a disjunction.

own(y,z)∧ wantstobuy(w,z)∧

(hasrequiredmoney(w,m)∨ hasgoodcreditrating(w))

—> cansell(y,w,z)

This rule is processed by the connectionist architecture shown in figure 3. We use the vectors $\mathbf{v_o}$,$\mathbf{v_w}$, $\mathbf{v_h}$ and $\mathbf{v_g}$ from the predicate tables *own*, *wantstobuy*, *hasrequiredmoney* and *hasgoodcreditrating* respectively. Though these are coarse-coded tuples their structure has the format of predicate code *p*, value of variable *1*, value of variable *2*,….value of variable *n* and predicate truth value *T / F*. These constituents are distinguishable and hence they can be used directly. These constituents are in coarse- coded form. We use these constituents to implement the complex rule under consideration. Component *z* is taken from both $\mathbf{v_o}$ and $\mathbf{v_w}$ and given as inputs to network 5.

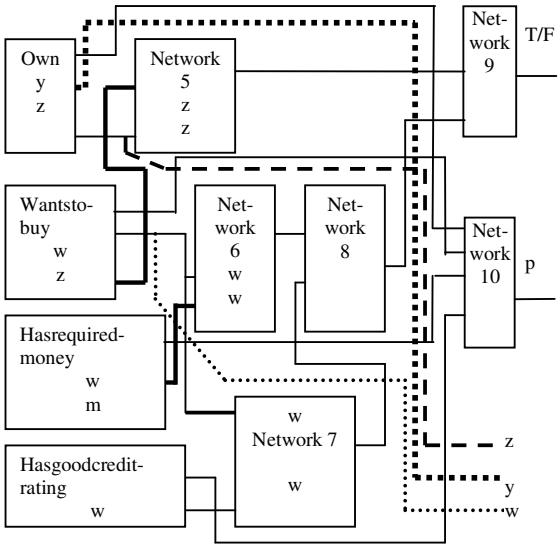

Fig. 3. Neural Networks for processing rule 3

This network generates truth value *T* or *F* depending on whether the values of variable *z* given to it are equal or unequal respectively. Similarly component w is taken both from vectors, $\mathbf{v_w}$ and $\mathbf{v_h}$

and given as inputs to network 6. This network generates truth value of *T* or *F* depending upon whether the values of variable *w* given to it are equal or unequal. Similarly component w is taken both from vectors, $\mathbf{v_w}$ and $\mathbf{v_g}$ and given as inputs to network 7. This network generates truth value of *T* or *F* depending upon whether the values of variable *w* given as inputs to it are equal or unequal. These truth values from outputs of network 6 and network 7 are given as inputs of network 8 which outputs *T* if either or both (inclusive or) of the truth values on its inputs are true else outputs *F*. The truth values from outputs of network 5 and network 8 are given as inputs to network 9 which outputs *T* if both of the truth values on its inputs are true else outputs *F*. The predicate code components of the vectors $\mathbf{v_o}$, $\mathbf{v_w}$, $\mathbf{v_h}$ and $\mathbf{v_g}$ are given as inputs to neural network 10 which outputs predicate code *p* for *cansell*. The values of *y, w* and *z* are passed on to the output lines as shown in figure 3 from the vectors $\mathbf{v_o}$, $\mathbf{v_w}$ and $\mathbf{v_o}$ respectively. If network 9 output is 'T' the values of *y, w* and *z* are accepted as belonging to vector $\mathbf{v_c}$ of the predicate table of *cansell.* Using this method we had processed a complex rule. The *variable binding problem* has been solved as described above while processing the above complex rule which is involving multiple conjunctions and a disjunction. The disjunction was meaning an inclusive OR operation. Our task was to check whether the variable *w* belonging to $\mathbf{v_w}$ and *either or both* of $\mathbf{v_h}$ and $\mathbf{v_g}$ are binding to same value. We had accomplished these with networks 6, 7 and 8 respectively. Similarly, we had to check whether variable *z* belonging to $\mathbf{v_o}$ and $\mathbf{v_w}$ are bound to same value. We had accomplished this with network 5.

We have solved the *variable binding problem* faced while implementing multiple conjunctions and a disjunction in a complex rule using *coarse-coded representations* without the need to decode them into localist representations. We have accomplished the variable binding task here using a *divide and conquer* strategy and distributed the total task to a set of neural networks which together accomplished the same. This approach enables us to perform easily variable binding in more complex rules which involve more number of conjunctions and disjunctions in them.

4 Testing

Following are the details of neural networks used to do the above mentioned work. The neural networks in table 5 are feed forward neural networks using back-propagation algorithm.

Table 5. Shows the details of neural networks used

Network	No. of input units	No.of hidden units	No.of output units
1	46	40	36
2	36	25	36
3	36	25	36
4	36	25	36
5	20	10	5
6	20	10	5
7	20	10	5
8	10	7	5
9	10	7	5
10	44	22	11
11	66	56	52
12	52	35	52

The reasoning task was successfully accomplished to give the expected results.

Table 6. Shows the details of tests 1 and 2

	No. of Training Patterns	No. of Test Patterns	No. of Patterns Corrected	No. of Patterns not Corrected
Localist reasoning system	216	108	60	48
	750	300	207	93
Coarse-coded reasoning system	216	108	89	19
	750	300	274	26

Secondly, the performance of the above coarse-coded reasoning system was compared for error tolerance under noise conditions with a localist representation based reasoning system (which was having identical number of input, hidden and output units for its neural networks). In the test 1, neural network 1 with 46 input units, 40 hidden units and 36 output units was trained with 216 patterns. 108 of these training patterns were made test patterns after introducing *1* bit error at a random location in each of these patterns. In the test 2, neural network 11 (implementing rule 1 for larger data) with 66 input units, 56 hidden units and 52 output units was trained with 750 patterns. 300 of these training

patterns were made test patterns after introducing *1* bit error at a random location in each pattern. Results are as shown in table 6. The coarse-coded reasoning system was found to be much more fault tolerant to errors compared to localist reasoning system as was indicated by tests performed.

In tests 3 to 6, the performance of Coarse-coaded reasoning system was compared with that of a Hopfield net for error correction tasks of same magnitude. In the tests 3 to 6, *neural network 12* having 52 input units, 35 hidden units and 52 output units was used. It implements auto-association of predicate *donate* with itself. It was trained with 250 coarse-coded patterns for auto-association of each pattern with itself. Some of the patterns were drawn from training set and converted to test patterns after introducing a 1bit error in each pattern at a random location to form the test set. These artificially introduced errors were simulating noise conditions. For each test pattern $\mathbf{v_{ts}}$ in the test set there is a corresponding pattern $\mathbf{v_{tr}}$ in the training set. *Network 12* was tested for its ability to generate the correct pattern $\mathbf{v_{tr}}$ on its outputs given the corresponding test pattern $\mathbf{v_{ts}}$ (with a single bit error) on its inputs. Results are as shown in table 7. For each of the tests conducted *Hopfield net* was designed to store all the patterns drawn from the training set which were used to obtain test patterns. Results of the tests indicate that the coarse-coded reasoning system exhibits better fault-tolerance than the Hopfield net for generating the correct pattern $\mathbf{v_{tr}}$ on its outputs given a test pattern $\mathbf{v_{ts}}$ on its inputs.

Table 7. Shows the details of tests 3, 4, 5 and 6

Test No.		No.of test patterns	No. of correct patterns generated
3	Coarse-coded reasoning System	100	78
3	Hopfield net	100	64
4	Coarse-coded reasoning System	150	117
4	Hopfield net	150	101
5	Coarse-coded reasoning System	200	155
5	Hopfield net	200	114
6	Coarse-coded reasoning System	225	172
6	Hopfield net	225	129

5 Conclusion

We have tested a connectionist forward chaining reasoning system using distributed coarse-coded representations on a given reasoning task. The system has successfully performed the given reasoning task. The coarse-coded reasoning system exhibited better fault tolerance over the localist reasoning system. The Coarse-coded reasoning system was found to be more fault-tolerant than a Hopfield net for the given error correcting tasks of the same magnitude. We have solved the variable binding problem faced in a novel way while implementing multiple conjunctions and a disjunction in a complex rule, using coarse-coded representations without the need to decode them into localist representations.

References:

[1] L. Shastri, Advances in SHRUTI: a neurally motivated model of relational knowledge representation and rapid inferencing using temporal synchrony, *Applied Intelligence,* 11(1), 1999, pp. 79-108.

[2] C. Wendelken and L. Shastri, Multiple instantiation and rule mediation in SHRUTI, *Connection Science*, 16, 2004, pp. 211-217.

[3] L. Shastri, C. Wendelken, Seeking coherent explanations --- a fusion of structured connectionism, temporal synchrony and evidential reasoning. *Proceedings of Cognitive Science 2000,* Philadelphia, PA, August 2000

[4] R. Sun, On variable binding in connectionist networks. *Connection Science*, 4, 1992, pp. 93-124.

[5] G.E. Hinton, J.L. McClelland and D.E. Rumelhart, Distributed representations. In D.E.Rumelhart and J.L.McClelland, editors, *Parallel Distributed Processing,* Vol.1. Cambridge, MA. MIT Press, 1986

[6] A. Browne, R. Sun, Connectionist inference models, *Neural Networks* 14, 2001, pp.1331-1355.

[7] A. Browne, R. Sun, Connectionist variable binding. *Expert systems: The International Journal of Knowledge Engineering and Neural Networks*, 16(3), 1999, pp. 189- 207.

[8] S. Haykins, *Neural Networks, a comprehensive foundation,* Second edition, New Jersey: Prentice hall, 1999

[9] S. Russel & P. Norvig, *Artificial Intelligence a Modern Approach,* Second Edition, Delhi: Pearson Education, 2003