Connectionist Reasoning System using Coarse-coded Distributed Representations

Sriram .G. Sanjeevi
Asst.Professor,
Dept. of Comp. Science & Engg.,
N.I.T. Warangal, Warangal 506004
91-0870-2430440
sgs@nitw.ernet.in

Dr. Pushpak Bhattacharya
Professor,
Dept. of Comp. Science & Engg.,
I.I.T. Bombay, Mumbai 400076
91-22-5767718
pb@cse.iitb.ac.in

ABSTRACT
In this paper, we describe a model for reasoning using forward chaining for predicate logic rules and facts with coarse-coded distributed representations for instantiated predicates in a connectionist frame work. Distributed representations are known to give advantages of good generalization, error correction and graceful degradation of performance under noise conditions. The system supports usage of complex rules which involve multiple conjunctions. The system solves the variable binding problem using coarse-coded distributed representations of instantiated predicates without the need to decode them into localist representations. The system has performed forward reasoning successfully on the given reasoning task. It's performance with regard to generalization on unseen inputs and its ability to exhibit fault tolerance under noise conditions is studied and has been found to give good results.

General Terms
Design, Reliability, Experimentation and Verification.

Keywords
Coarse-coding, distributed representations, forward chaining, predicates, connectionist frame work, fault tolerance.

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 [3] 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], [5] 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* [2] 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 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 it's instantiated predicates.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Conference'04, Month 1–2, 2004, City, State, Country.
Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

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) — > *cansell(y,w,z);*
5 *give(John,Mary,Book-17);*
6 *buy(Chris,Book-1);*
7 *wantstobuy(Walter,Book-1);*
8 *hasrequiredmoney(Walter,Money-1);*

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

1. own(Mary,Book-17);

2. donate(Mary,Book-17);

3. own(Chris,Book-1);

4. cansell(Chris,Walter,Book-1);

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 a 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 predicate fact. The size of each k is chosen for a $\mathbf{v_{ij}}$ depending on the requirements of predicate p_i. 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 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

3.1 Connectionist Reasoning Using Localist Representations

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 connectionst 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 otherwise 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* of rule 4 also get activated then rule 4 fires.

3. own(y,z) —> donate(y,z) and

4. own(y,z) ^ wantstobuy(w,z) ^ hasrequiredmoney(w,m)

—>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 the samples of localist tuples used by *give*

S.No.	Predicate *id* code	Localist Value of *x*	Localist Value of *y*	Localist Value of *z*	Truth Value of Predicate
1	0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 1
215	0 0 0 0 1 0 0 0 0 0 0	0 0 0 0 1 0 0 0 0 0	0 0 0 0 1 0 0 0 0 0	0 0 0 0 0 1 0 0 0 0	0 0 0 0 1

Table 2 Shows the sample of localist tuples used by predicate *Cansell*

S.No	Predicate *id* code	Localist Value of *y*	Localist Value of *w*	Localist Value of *z*	Truth Value of Predicate
5	0 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 1	0 0 0 0 0 1 0 0 0 0	0 0 0 0 1
46	0 0 0 0 0 0 0 1 0 0 0	0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 1 0	0 0 0 0 0 0 1 0 0 0	0 0 0 0 1

3.2 Obtaining Coarse-coded Distributed Representations from Localist Representations

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

0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1

The first binary string of 11 bits denotes predicate code, the next 10 bits denotes the value being assigned to variable *x,* the next 10 bits the value for variable *y*, the next 10 bits the value for variable *z* and the last 5 bits indicate the truth value of predicate. These localist tuples need to be converted into coarse-coded representations to be used by our reasoning system. We explain here the process of obtaining coarse-coded representation for the above localist vector. We view the above vector as being kept in overlapping coarse zones of length of 4 consecutive bits. We encode the zone as *1* if there is atleast one *1* bit in that zone or else as *0*. We then consider next coarse zone and encode it as *1* or *0* following above method. We do this process left to right starting from the left most bit. That means first coarse zone will have first *4* bits in its coarse zone. The second coarse zone will have bits *2* to *5* in its coarse zone. We do this encoding process for above localist tuple to yield following coarse-coded tuple

01111000000 0000001111 0000011110 0000011110 01111. We can easily obtain the original localist pattern by

replacing the substring '1111' with '0001' in the coarse-coded pattern. 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.

3.2.1 Advantages of coarse-coding

Coarse-coding increases the information capacity by increasing the number of units active at a time compared to localist codes which have sparsely populated 1's. The amount of information [4] conveyed by a unit that has a probability *p* of being '*1'* is

– plog (p) – (1 - p)log(1 - p).

Because of higher information capacity it supports features like generalization, fault tolerance and graceful degradation of performance under noise conditions. For this application of predicate calculus reasoning problem coarse-coding the predicate and argument representations is expected to give above advantages. In the following part of the work, we observe and note the generalization, fault tolerance and error correction capabilities obtained when coarse-coding is used in feed-forward connectionist architectures in a forward chaining connectionist reasoning system.

In the previous section, we have shown the samples of localist representations of the tuples of some of the predicates. We show here the coarse-coded representations of the predicates in rule base.

Table 3 Shows a sample of coarse-code representation of data tuples used by predicate *Give*

S.No of Tuple	Predicate 'id' code	Localist Value of *x*	Localist Value of *y*	Localist Value of *z*	Truth Value of Predicate
Tuple no. 215	0 1 1 1 1 0 0 0 0 0 0	0 1 1 1 1 0 0 0 0 0	0 1 1 1 1 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0	0 1 1 1 1

Table 4 Shows a sample of coarse-code representation of data tuples used by predicate *Own*

S.No of Tuple	Predicate 'id' code	Localist Value of *x*	Localist Value of *y*	Truth Value of Predicate
Tuple no. 35	0 0 1 1 1 1 0 0 0 0 0	0 1 1 1 1 0 0 0 0 0	0 0 1 1 1 1 0 0 0 0	0 1 1 1 1

Table 5 Shows a sample of coarse-code representation of data tuples used by predicate *Buy*

S.No of Tuple	Predicate 'id' code	Localist Value of *x*	Localist Value of *y*	Truth Value of Predicate
Tuple no. 34	1 1 1 1 0 0 0 0 0 0 0	0 1 1 1 1 0 0 0 0 0	0 0 0 1 1 1 1 0 0 0	0 1 1 1 1

Table 6 Shows a sample of coarse-code representation of data tuples used by predicate *wantstobuy*

S.No of Tuple	Predicate 'id' code	Localist Value of *x*	Localist Value of *y*	Truth Value of Predicate
Tuple no. 33	0 0 0 0 0 1 1 1 1 0 0	0 1 1 1 1 0 0 0 0 0	0 0 0 0 1 1 1 1 0 0	0 1 1 1 1

Table 7 Shows a sample of coarse-code representation of data tuples used by predicate *hasrequiredmoney*

S.No of Tuple	Predicate 'id' code	Localist Value of *x*	Localist Value of *y*	Truth Value of Predicate
Tuple no. 31	0 0 1 1 1 1 0 0 0 0 0	0 1 1 1 1 0 0 0 0 0	0 0 0 0 0 0 1 1 1 1	0 1 1 1 1

Table 8 Shows a sample of coarse-code representation of data tuples used by predicate *Cansell*

S.No of Tuple	Predicate 'id' code	Localist Value of *y*	Localist Value of *w*	Localist Value of *z*	Truth Value of Predicate
Tuple no. 46	0 0 0 0 1 1 1 1 0 0 0	0 0 0 0 0 1 1 1 1 0	0 0 0 0 0 1 1 1 1 0	0 0 0 1 1 1 1 0 0 0	0 1 1 1 1

3.3 Organization of Neural Networks in the Connectionist Reasoning System

The neural networks shown in the following diagrams accomplish the forward reasoning using above 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 v_i from the predicate table *give* the network 1 generates on its outputs a vector v_o from the predicate table own. This in turn impresses on the inputs of network 2 to generate a vector v_d on its outputs. These vectors are in coarse-coded form and denote a predicate fact. Hence impressing a vector v_i denoting a predicate fact on the inputs of network 1 has generated a vector v_o on its outputs denoting another predicate fact from the predicate table *own*. This way the rule *give*(*x,y,z*) —> *own*(*y,z*) was processed. The impression of v_o on the inputs of network 2 generated a vector v_d on its outputs denoting a predicate fact from the predicate table *donate*. This processes the rule *own(y,z)* —> *donate (y,z).* So here we see the rules 1 and 3 are getting activated in a forward chaining fashion . This is said to accomplish forward reasoning .

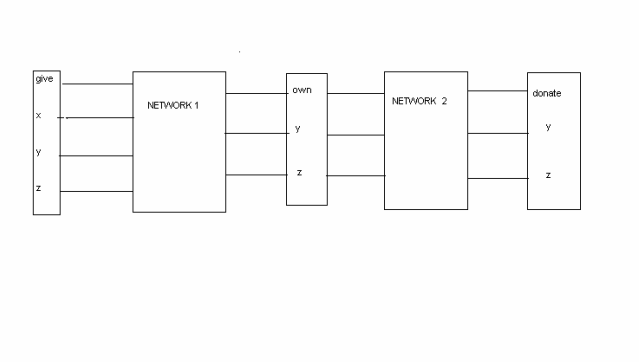

Figure 1. Neural Networks for processing rules 1 and 3

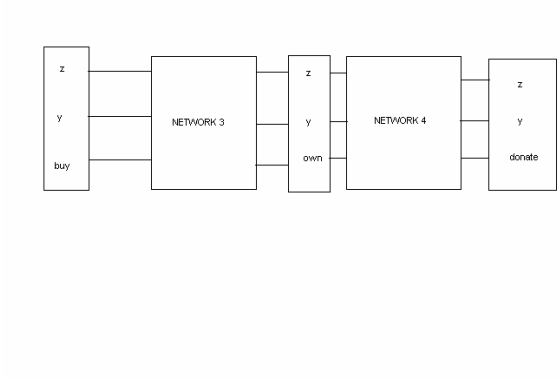

Figure 2. Neural Networks for processing rules 2 and 3

Similarly impressing a vector $\boldsymbol{v_b}$ on the inputs of network 3 generates vector $\boldsymbol{v_o}$ on its outputs and this in turn gets impressed on inputs of network 4 generating on its outputs vector v_{dv} 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.

3.4 Variable Binding during Processing of the Complex Rule

Consider the complex rule which involves multiple conjunctions.

own(y,z) ^ *wantstobuy(w,z)* ^ *hasrequiredmoney*(w,m) —>

cansell(y,w,z).

This rule is processed by the connectionst architecture shown in figure 3. We use the vectors $\mathbf{v_o}$,$\mathbf{v_w}$ and $\mathbf{v_h}$ from the predicate tables *own, wantstobuy* and *hasrequiredmoney* respectively. Though these are coarse-coded tuples their structure has the format of predicate code p, value of variable *1*, variable *2* 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 $\boldsymbol{v_o}$ and $\boldsymbol{v_w}$ and given to network 5. This network generates truth value *T* or *F* depending on whether the values of variable *z* given to it are same or different. Similarly component w is taken both from vectors, $\mathbf{v_w}$ and $\mathbf{v_h}$ and given to network 6. This network generates truth value of *T* or *F* depending upon whether the values of variable *w* given to it are same or different. These truth values from network 5 and network 6 outputs are given to network 7 which outputs *T* if both of the truth values on its inputs are true else outputs *F*. The predicate code components of the vectors are given to another neural network *8* 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 $\mathbf{v_h}$ respectively. If network 7 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 the variable binding problem has been solved while processing the above complex rule which is involving multiple conjunctions. Our task was to check whether the variable *w* belonging to both *wantstobuy* and *hasrequiredmoney* are binding to same value. Similarly, we had to check whether variable z belonging to *own* and *wantstobuy* are bound to same value. We had accomplished these with networks 6 and 5 respectively.

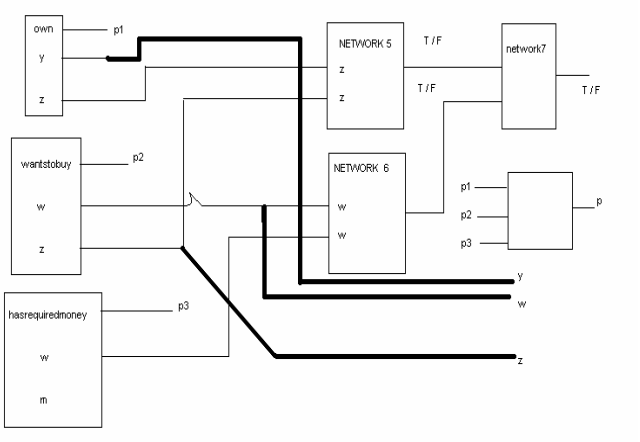

Figure 3. Neural Networks for processing rule 4

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 can be similarly extended to handle even more complex rules which involve more number of conjunctions.

4. TESTING

Following are the details of neural networks used to do above mentioned work.

Table 9 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	15	10	5
8	33	25	11
9	66	56	52

The neural networks in table *9* are feed forward neural networks using back-propagation algorithm. The reasoning task was successfully accomplished to give the expected results. 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). Tests were performed using the SNNS simulator. In

the test 1, neural network 1 was trained with 216 patterns. A subset 108 of them were made test patterns after introducing 1 bit error at a random location in each pattern.

Table 10 Shows the details of test 1

Test 1	No. of training patterns	No.of test patterns	No.of patterns corrected	No.of patterns not corrected
Localist reasoning system	216	108	60	48
Coarse-coded reasoning system	216	108	89	19

In the test 2, neural network 9 with 66 input units, 56 hidden units and 52 output units was trained with 750 patterns. The performance of the above for coarse-coded patterns was compared for error tolerance under noise conditions with another network (which was having identical number of input, hidden and output units) but which used localist patterns. A subset 300 of the training patterns were made test patterns after introducing 1 bit error at a random location in each pattern.

Table 11 Shows the details of test 2

Test 2	No. of training patterns	No.of test patterns	No.of patterns corrected	No.of patterns not corrected
Localist reasoning system	750	300	207	93
Coarse-coded reasoning system	750	300	274	26

In tests 3 and 4, Neural network 9 was tested with coarse-coded patterns for generalization on unseen test patterns after completing the training with a training set.

Table 12 Shows the details of test 3

Test 3	No. of training patterns	No.of unseen test patterns	No.of patterns correctly generalized	No.of patterns not correctly generalized
Coarse-coded reasoning system	650	350	342	8

Table 13 Shows the details of test 4

Test 4	No. of training patterns	No.of unseen test patterns	No.of patterns correctly generalized	No.of patterns not correctly generalized
Coarse-coded reasoning system	700	300	300	0

5. CONCLUSIONS

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 system has displayed good generalization ability on unseen test patterns. 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. These artificially introduced errors were simulating noise conditions. We have also solved the variable binding problem faced while implementing multiple conjunctions(in a complex rule) using coarse-coded representations without the need to decode them into localist representation.

6. REFERENCES

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

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

[3] T.J. Van Gelder, Defining ‘ distributed representation’, *Connection Science* 4 (3 and 4) (1992) 175-192.

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

[5] Shastri, L. and V. Ajjanagadde. (1993) From simple associations to systematic reasoning: A connectionist representation of rules, variables and dynamic bindings. *Behavioral and Brain Sciences,* 16(3), 417-494.

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

[7] A. Browne, R.Sun. Connectionist inference models, *Neural Networks* 14 (2001) 1331-1355

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

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