Connectionist Predicate Logic Model with Parallel Execution of Rule Chains

S G Sanjeevi
Dept. of Comp. Science & Engineering
N.I.T. Warangal,
Warangal, India
+91-870-2462711
sgsanjeevi@yahoo.com

P Bhattacharyya
Dept. of Comp. Science & Engineering
IIT 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 and disjunctions. The system supports parallel and independent execution of predicate logic rule chains in a connectionist environment. The system solves the variable binding problem in a new way using coarse-coded distributed representations of instantiated predicates. Its 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.

Categories and Subject Descriptors

I.5.1 [**Pattern Recognition**]: Models – *neural nets*.

General Terms

Design, Reliability, Experimentation.

Keywords

Coarse-Coding, Connectionist, Parallel Rule Chains, Reasoning, 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 [2] 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* [6] described a localist method where temporal synchrony was used to create bindings between variables and entities they represent. *CONSYDERR* [1] described another localist method for variable binding and forward reasoning. Since, these systems used localist representations, advantages of distributed representations are not obtainable by them and hence the motivation for a distributed representation based reasoning system. In our previous works, we have designed and developed connectionist reasoning systems which use distributed representations [3, 4, 5].

In this present work, we describe the design and implementation of a connectionist reasoning system which uses coarse-coded distributed representations and supports parallel execution of predicate logic rule chains. The organization of the paper is as follows. In section 2, we describe the rules and facts base. In section 3, we explain the connectionist reasoning using forward chaining of rules and in section 4 we describe obtaining the coarse-coded representations. In section 5, we describe connectionist reasoning system and in section 6 parallel execution of rule chains. In section 7, we describe testing and results and in section 8 conclusions.

2. RULES AND FACTS BASE

Our system represents and reasons with the following predicate logic rules and facts:

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

Using the above knowledge base, some of the inferences made by the system are shown below.

1. own(Mary, Book-1);

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

Our objective is to start with the above knowledge base and obtain the results of inference correctly by the coarse-coded connectionist reasoning system. First, we describe how inference is performed in a localist model of reasoning using neural networks. This is because, we start with localist representations of predicates instantiated with arguments and then later convert them to coarse-coded distributed representations.

3. CONNECTIONIST FORWARD REASONING

We describe here, briefly with an example how forward reasoning using localist representations [1], [6] is made using a connectionist system. Let us consider the *rule 1: give(x,y,z)—> own(y,z)* from the knowledge base. The localist pattern for the LHS of rule 1 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*, '*John'*, the next 3 bit value denotes an object getting bound to variable *y* , *'Mary'* and the next value denotes, '*Book-1'*. The last bit indicates the truth value of predicate *give*. This instantiation 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. Because of the rule activation the localist pattern representation for RHS will be *0010 001 001 1* denoting *own(Mary,Book-1)*. This triggers the rules whose left hand sides match RHS of rule *1* and through this forward chaining, forward reasoning using localist representations is accomplished. The binding information is similarly passed on in these rules for the variables. In this manner, forward reasoning is accomplished using localist representations in a connectionist reasoning system.

4. OBTAINING COARSE-CODED REPRESENTATIONS

4.1 Localist Representation Vectors for the Instantiated Predicates

In the knowledge base, there are *8* predicates. They are *give, buy, own, candonate, wantstobuy, hasrequiredmoney, hasgoodcreditrating* and *cansell.* We need separate predicate codes to represent these predicates. We represent each of these predicates by a separate code which is the localist representation for its predicate identification code. In *Table 1* below we show a sample of localist vectors used for the predicate *give* 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

4.2 Coarse-Coded Representation Vectors for the Instantiated Predicates

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 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 increases the information capacity [2] 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 *– plog (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 in table 2 a sample of coarse-coded representation of the tuples for predicate *give* in the rule base.

Table 2. 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

5. CONNECTIONIST REASONING SYSTEM

5.1 Organization of Neural Networks

The neural networks shown 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.

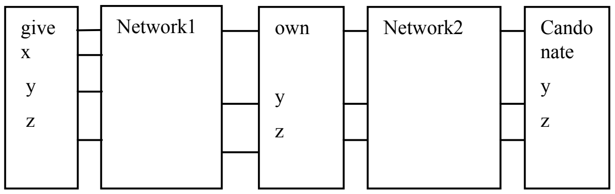

Figure 1. Neural Networks for processing rules *(1)* and *(3)*

When impressed on its inputs with one of the vectors v_g from the predicate table *give* the network 1 generates on its outputs a vector 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) → candonate(y, z)*. These vectors are in coarse-coded form and denote a predicate fact. So we see the rules *1* and *3* are getting activated in a forward chaining fashion. Similarly, rules *(2), (3)* are processed by neural networks 3 and 4. Neural networks *1, 2* and *3, 4* execute the following rule chains (I), (II) respectively.

The rule chains (I), (II) are executed independently.

● *give(x, y, z) → own(y, z) → candonate(y, z)* (I)

● *buy(y, z) → own(y, z) → candonate(y, z)* (II)

5.2 Variable Binding during Processing of Complex Rule having Multiple Conjunctions and a Disjunction

We describe here, how variable binding is done in a complex predicate logic rule, having both conjunctions and a disjunction. Consider, the following complex rule which is the rule 4 in our knowledge base which involves multiple conjunctions and a disjunction:

● *own(y, z)∧wantstobuy(w, z)∧(*hasr*equiredmoney(w, m) ∨ hasgoodcreditrating(w)) → cansell(y, w, z).* (4)

This rule is having a disjunction which denotes an *inclusive OR* operation. The rule is processed by the connectionist architecture shown in figure 2.

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. These are coarse-coded vectors and their structure has the format of predicate code *p*, value of variable *1*, value of variable *2,…,* value of variable *n* and the predicate truth value *T/F.* These constituents of the vectors are distinguishable and hence they can be used directly. These constituents are present in the coarse-coded form. We use these vectors $\mathbf{v}_o$, $\mathbf{v}_w$, $\mathbf{v}_h$ and $\mathbf{v}_g$ to implement the complex rule under consideration.

Component *z* is taken from both the vectors $\mathbf{v}_o$ and $\mathbf{v}_w$ and given as inputs to neural network *5.* This network generates as output truth value *T* or *F* depending on the values of variable *z* given to it as inputs are *equal* or *unequal* respectively. Similarly, component *w* is taken from both the vectors $\mathbf{v}_w$ and $\mathbf{v}_h$ and given as inputs to the neural network *6.* This network generates output truth value of *T* or *F* depending upon the values of variable *w* given to it as inputs are *equal* or *unequal* respectively. Similarly, component *w* is taken from both the vectors, $\mathbf{v}_w$ and $\mathbf{v}_g$ and given as inputs to the neural network *7.* This network generates as output truth value of *T* or *F* depending upon the values of variable *w* given as inputs to it are equal or unequal respectively. The truth values from the outputs of neural network *6* and neural network *7* are given as inputs to the neural network *8.* Neural network *8* outputs *T,* if either or both *(inclusive or)* of its inputs are having truth value *T* else outputs *F.* The truth values from outputs of neural network *5* and neural network *8* are given as inputs to the neural network *9.* Neural network *9* outputs *T* if both of its inputs have truth value *T* 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 the 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 2 from the vectors $\mathbf{v}_o$, $\mathbf{v}_w$ and $\mathbf{v}_o$ respectively. If neural network *9's* 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.*

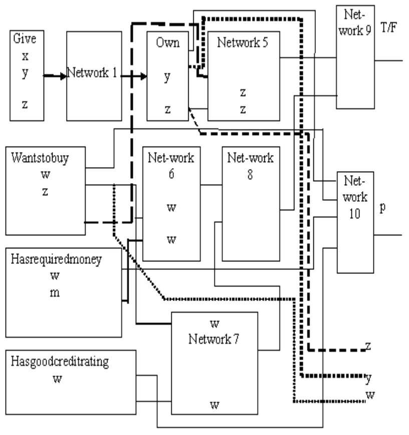

Figure 2. Neural Networks for processing the rule chain shown in (III)

Using this method, we had processed the complex rule having two conjunctions and a disjunction. The *variable binding problem* has been solved as described above, while processing the complex rule which is involving multiple conjunctions and a disjunction. We used a *divide and conquer* approach and distributed the total variable binding 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.

6. PARALLEL EXECUTION OF RULE CHAINS

The rule *(4)* which involves multiple conjunctions and a disjunction is part of the two rule chains shown below in (III) and (IV).

• *give(x, y, z) → own(y, z) ∧ wantstobuy(w, z) ∧(hasrequiredmoney(w, m)∨ hasgoodcreditrating(w)) → cansell(y, w, z);* (III)

• *buy(y, z) → own(y, z) ∧ wantstobuy(w, z) ∧(hasrequiredmoney(w, m) ∨ hasgoodcreditrating(w)) → cansell(y, w, z);* (IV)

The connectionist architecture shown in figure 2 has processed the rule chain shown in (III). Since, rule *4* is part of two different rule chains, the connectionist architectures for processing the rule in these two rule chains will be identical but separate and they will be processing the rule independently of each other. The connectionist architecture for processing the rule chain shown in (IV) is shown in figure 3. Neural Networks 5A, 6A, 7A, 8A, 9A and 10A shown in figure 3 have the same architecture as neural networks *5, 6, 7, 8, 9* and *10* shown in figure 2 respectively. They will be processing the rule *4* in an identical way as neural networks *5, 6, 7, 8, 9* and *10* respectively. Rule chains shown in (III) and (IV) are processed separately by the connectionist architectures shown in figures 2 and 3 respectively. Hence, these rule chains can be executed in parallel and independently.

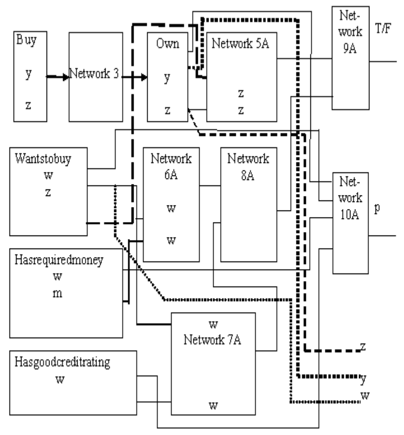

Figure 3. Neural Networks for processing the rule chain shown in (IV)

7. TESTING

Reasoning task was successfully accomplished to give expected inference results using the rules and facts shown in section 2. 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).

Table 3. Shows the details of test1

Test 1	No. of training patterns	No.of test patterns	No.of output patterns correctly generated
Localist reasoning system	750	300	207
Coarse-coded reasoning system	750	300	274

In the test 1, 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 of the input patterns leaving the output patterns unchanged. If output pattern is correctly produced by the *neural network 11* despite having a *1* bit error in the input side of the test pattern, it indicates the fault tolerance property of the connectionist system. Results are as shown in table 3. In tests 2 and 3, Neural network 11 was tested with coarse-coded patterns for generalization on unseen test patterns as shown in table 4.

Table 4. Shows the details of tests 2 and 3

Tests	Training Patterns	Unseen Test Patterns	Correctly Generalized	Not Correctly Generalized
2	650	350	342	8
3	700	300	300	0

8. CONCLUSIONS

We have designed and 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. 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. The system supported parallel predicate logic rule chains in a connectionist environment. These rule chains were executed in a parallel and independent way. Thus, system supports parallel processing of rule chains. 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. The system has displayed good generalization ability on unseen test patterns.

9. REFERENCES

[1] Browne, A., and Sun, R. Connectionist inference models. Neural Networks. 14: 2001, 1331-1355.

[2] Hinton, G.E., McClelland, J.L., and Rumelhart, D.E. Distributed representations. Parallel Distributed Processing: Explorations in Microstructure of Cognition. 1986, 1:77-109, , MIT Press, Cambridge.

[3] Sanjeevi, S.G., and Bhattacharyya, P., A Connectionist Model for Predicate Logic Reasoning using Coarse-coded Distributed Representations. In Proceedings of the 9th International conference on Knowledge-based & Intelligent Information and Engineering Systems(KES 2005), Melbourne, Australia, Lecture Notes in Artificial Intelligence, Springer-Verlag, Berlin Heidelberg, No. 3682, 732-738.

[4] Sanjeevi, S.G., and Bhattacharyya, P. Non-Monotonic reasoning with connectionist networks using coarse-coded Representations. In Proceedings of the 5th IEEE International conference on Machine Learning and Cybernetics (Dalian, China, August 13th-16th, 2006). Vol. 5, pp. 3048-3052.

[5] Sanjeevi, S.G., and Bhattacharyya, P., A fault tolerant connectionist model for predicate logic reasoning, variable binding: using coarse-coded distributed representations. WSEAS Transactions on Systems, 4, 4, (Apr. 2005), 331-336.

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