A Novel Modular Neural Architecture for Rule-Based and Similarity-Based Reasoning

Rafal Bogacz and Christophe Giraud-Carrier

Department of Computer Science, University of Bristol
Merchant Venturers Building, Woodland Rd
Bristol BS8 1UB, UK
{bogacz,cgc}@cs.bris.ac.uk

Abstract. Hybrid connectionist symbolic systems have been the subject of much recent research in AI. By focusing on the implementation of high-level human cognitive processes (e.g., rule-based inference) on low-level, brain-like structures (e.g., neural networks), hybrid systems inherit both the efficiency of connectionism and the comprehensibility of symbolism. This paper presents the Basic Reasoning Applicator Implemented as a Neural Network (BRAINN). Inspired by the columnar organisation of the human neocortex, BRAINN's architecture consists of a large hexagonal network of Hopfield nets, which encodes and processes knowledge from both rules and relations. BRAINN supports both rule-based reasoning and similarity-based reasoning. Empirical results demonstrate promise.

1 Introduction

Over the past few years, the mainly historical, and arguably unproductive, division between psychological and biological plausibility has narrowed significantly through the design and implementation of successful hybrid connectionist symbolic systems. Rather than committing to a single philosophy, such systems draw on the strengths of both biology and psychology, by implementing high-level human cognitive processes (e.g., rule-based inference) within low-level, brain-like structures (e.g., neural networks). Hence, hybrid systems inherit the characteristics of both traditional symbolic systems (e.g., expert systems) and connectionist architectures, including:

- Complex reasoning
- Learning and generalisation from experience
- Efficiency through massive parallelism

This paper presents the Basic Reasoning Applicator Implemented as a Neural Network (BRAINN). The original description of BRAINN is in [1]. The architecture of BRAINN mimics the columnar organisation of the human neocortex. It consists of a large hexagonal network of Hopfield nets [7] in which both rules and relations can be encoded in a distributed fashion. Each relation is stored in a single Hopfield net, whilst each rule is stored in a set of adjacent Hopfield nets. Through systematically orchestrated relaxations, BRAINN combines

S. Wermter and R. Sun (Eds.): Hybrid Neural Systems, LNAI 1778, pp. 63–77, 2000.

rule-based reasoning typical of expert systems with similarity-based reasoning typical of neural networks. Hence, BRAINN supports both monotonic reasoning and several forms of common-sense (non-monotonic) reasoning [10].

The paper is organised as follows. Section 2 details the BRAINN architecture and algorithms. Section 3 reports the results of a number of experiments with BRAINN. Section 4 reviews related work, and section 5 concludes the paper and outlines directions for future work.

2 BRAINN

BRAINN's architecture is inspired by the columnar organisation of the human neocortex [3]. The human neocortex is divided into minicolumns (Ø0.03mm), i.e., groups of neurons gathered around dendrite bundles. Minicolumns create a hexagonal network, and are, in turn, organised in hexagonal macrocolumns (Ø0.5mm). The axons of some pyramidal neurons have an unusually high number of synapses within about 0.5mm of their soma. Simultaneous activation of neurons, in a 0.5mm radius, is thus sometimes observed. These neurons excite one another and can in turn recruit additional neurons, since the adjacent neurons receive activation from two or more neurons, as shown in Figure 1.

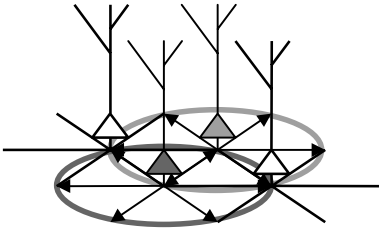

Fig. 1. Neuron Recruitment

Calvin [3] speculates further that newly recruited neurons can subsequently recruit others, so that the pattern of activation of one or two macrocolumns can spread through the whole network. He argues that this process is especially relevant to short term memory phenomena.

2.1 Knowledge Implementation

BRAINN's underlying neural network architecture supports the encoding of both relations and if-then rules, as detailed in the following sections.

Relations The relations considered here can be represented as triples of the form <Object, Attribute, Value>. In BRAINN, each component of the triple is represented by a unique pattern. The pattern is a sequence of bits of constant

length N, where each bit may have value -1 or +1. Hence, each relation can be stored in a Hopfield network with $3N$ units. Figure 2 shows one such network for $N = 4$. The activations of units x_1 to x_N correspond to the binary representation of the object, the activations of units x_{N+1} to x_{2N} correspond to the binary representation of the attribute and the activations of units x_{2N+1} to x_{3N} correspond to the binary representation of the value.

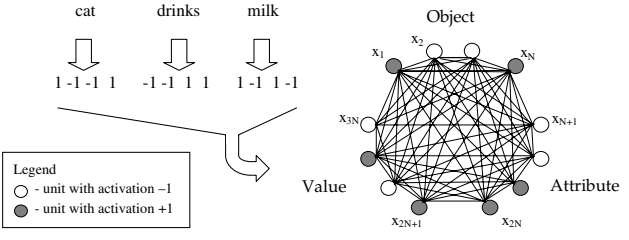

Fig. 2. Relation Encoding

For simplicity, Hopfield networks encoding relations are referred to as assemblies, which work as associative memories. After delivering two components of a relation to an assembly, the third one may be retrieved. The weights of the network are set using either the Hebb rule or the Perceptron rule [5], as detailed below.

Weights Setting with the Hebb Rule. Upon creation of the network, all weights are initialised to 0. For each new triple to remember, the binary representations of its components are delivered to the corresponding neurons in the assembly as shown in Figure 2. Then, the weights w_{ij} between units i and j ($i \neq j$) are updated according to equation 1.

$$w_{ij} \leftarrow w_{ij} + x_i x_j \tag{1}$$

Weights Setting with the Perceptron Rule. The Perceptron rule is similar to the Hebb rule, but it increases storage capacity and reduces susceptibility to pattern correlation [5]. With the Perceptron rule, the weights w_{ij} to unit i are modified according to equation 1 only if the output x_i^{t+1} of unit i (as per equation 2 below), is different from the bit value in the triple's representation x_i^t. In addition, learning is iterative. Patterns are presented repeatedly until they are stored correctly or the learning time exceeds a pre-defined limit. The learning algorithm is shown in Figure 3.

The network functions as an associative memory, using the principle of relaxation to retrieve relations. A pattern is delivered to the network and, during relaxation, unit i changes its state x_i according to equation 2 until the network reaches a stable state.

$$x_i^{t+1} = sgn(\sum_{j=1}^{3N} w_{ij} x_i^t) \tag{2}$$

Initialise all weights to 0
Repeat

– For each triple to remember
  1. Deliver triple's elements to the network's units x_i
  2. For each unit i
     (a) Compute activations x_i^{t+1}
     (b) If $x_i^{t+1} \neq x_i^t$ Then update weights: $w_{ij} \leftarrow w_{ij} + x_i^t x_j^t (j \neq i)$

Until no updates are made or time is out

Fig. 3. Weight Setting with Perceptron Rule

The Hopfield network stabilises on the stored pattern most similar to the delivered one or sometimes in a random state called a spurious attractor. In BRAINN, all of the questions asked by the user take the form of a triple <Object, Attribute, Value>, where one component is replaced by a question mark (e.g., <mouse, eats, ?>). The network then retrieves the triple's missing component based on knowledge of the other two. The units corresponding to the unknown component are set to 0. If the question is delivered to an assembly remembering the expected relation, then, after relaxation, the units corresponding to the unknown component are equal to the binary representation of the relation's missing element. If the network does not store the triple, it stabilises in a spurious attractor or another remembered triple. Examples are in section 2.3.

Given the above mapping of relations to triples, it is possible that an object may have more than one value for a single attribute, e.g., <cat, eats, whiskas> and <cat, eats, mouse>. To solve this problem, such triples are stored in distinct assemblies. Details are described in section 2.4.

If-Then Rules The rules that BRAINN uses are traditional if-then rules, where the left-hand side (LHS) consists of a conjunction of conditions and the right-hand side (RHS) is a single condition, for example:

```
IF <soil, is, sandy> AND <soil, humus_level, high>
THEN <soil, compaction, high>
```

As with relations, conditions are represented by triples of the form <Object, Attribute, Value> and subsequently stored in assemblies of the form described in section 2.1.1. The various assemblies representing the conditions of a rule can then be connected into a network of assemblies that encodes the rule. Figure 4 shows such a network for the above rule. In the network, there are connections between each unit from the LHS assemblies and each unit from the RHS assembly. The weights between assemblies are set according to the Hebb rule. Therefore, if one knows one side of the rule, one can retrieve the other.

To accommodate rules with varying numbers of conditions in LHS and to provide a uniform network topology for both rules and relations (rather than a set of disconnected networks), assemblies are organised into a large hexagonal

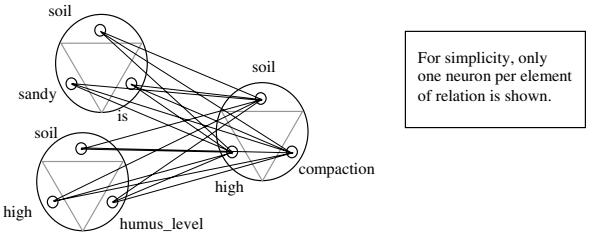

Fig. 4. Rule Encoding in Network

network as shown in Figure 5. With such an architecture, each rule may have a maximum of 6 conditions in its LHS. Larger numbers of conditions in LHS can, of course, be handled by chaining rules appropriately, using new variables (e.g., IF A AND B AND C THEN D can be rewritten as IF A AND B THEN E, and IF E AND C THEN D).

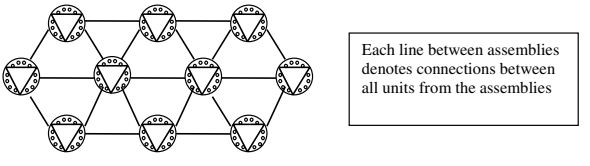

Fig. 5. Hexagonal Network of Assemblies

When reasoning with rules, BRAINN implements backward chaining. Hence, the network retrieves the LHS of a rule upon delivery of its RHS. The following, along with Figure 6, details how this takes place in the network. For the sake of argument, assume that a rule consists of four conditions in its LHS. The RHS is stored in one assembly and the four conditions from the LHS are stored in adjacent assemblies. Upon activation of the assembly corresponding to the RHS, the network must retrieve the four conditions of the LHS.

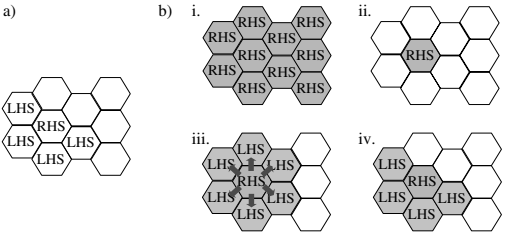

Fig. 6. Rule Retrieval: a) Storage; b) Retrieval - i) RHS sent to all assemblies, ii) Relaxation, iii) LHS sent to adajacent assemblies and iv) Relaxation

First, the RHS is delivered to all the assemblies in the hexagonal network. Once all of the Hopfield networks have relaxed, only the assembly storing the RHS has stabilised on the delivered pattern, since for that assembly, delivered and stored patterns are the same. Then, the assembly storing the RHS sends its pattern (vector of activation) to all six adjacent assemblies. Each adjacent assembly receives the pattern vector of the RHS assembly multiplied by the matrix of weights between the RHS assembly and itself. All of the adjacent assemblies relax after receiving the vector. In the case of the four LHS assemblies, the vector received is one of the patterns remembered in the local Hopfield network, so these assemblies will be stable. The other two assemblies will not be stable and will thus change their state. Moreover, the four LHS assemblies now send their patterns back to the RHS assembly. The RHS assembly receives these patterns multiplied by the matrix of weights between the LHS assemblies and itself. Thus, the pattern received by the RHS assembly is equal to its own pattern of activation. A kind of resonance is achieved, allowing the retrieval of the correct LHS.

Note that LHS assemblies recruited by the aforementioned process are implicitly conjoined, i.e., the left-hand side of the rule is the conjunction of the conditions found in all of the LHS assemblies retrieved. To avoid confusion during backward chaining when several rules have the same right-hand sides (e.g., IF A AND B THEN C, and IF D THEN C), the right-hand sides are stored in different assemblies.

Rules with Variables The rules discussed so far are essentially propositional. It is often useful, and even necessary, to encode and use more general rules, which include variables. To reason in the presence of such rules, an effective way of binding variables is required. In BRAINN, variable binding is achieved by using special weight values between LHS and RHS assemblies, as shown in Figure 7 for the rule IF <&someone, drinks, milk> THEN <&someone, is, strong>. Let $\&X$ be the variable. Then, the weights between the units representing $\&X$ in LHS and the units representing $\&X$ in RHS are equal to 1, whilst the weights between the units representing $\&X$ and all other units are equal to 0.

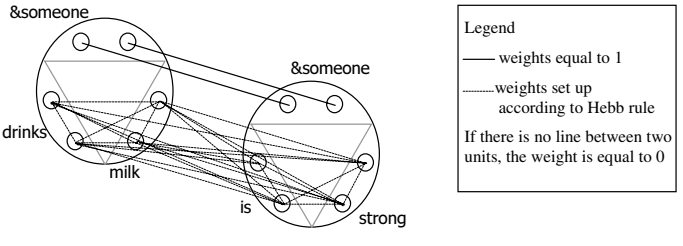

Fig. 7. Weight Setting for a Simple Rule

With such a set of weights, the pattern for the variable is sent between assemblies without any modifications nor interactions with the rest of the information in the assembly. The weights inside the LHS assemblies and the RHS assembly must also satisfy similar conditions. That is, the weight of self-connection for all units representing a variable is equal to 1, whilst the weight between each unit representing a variable and any other unit is equal to 0. These latter conditions guarantee the stability of the assembly, which is critical to the reasoning algorithm.

2.2 Functional Overview

Although BRAINN's knowledge implementation is inspired by biological considerations, its information processing mechanisms are not biologically plausible. A high-level view of BRAINN's overall architecture is shown in Figure 8.

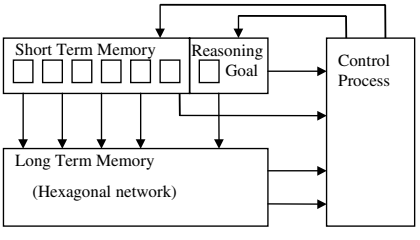

Fig. 8. BRAINN's Architecture

The system's knowledge (i.e., rules and relations) is stored in the Long Term Memory (LTM). Temporary, run-rime information is stored in the Short Term Memory (STM) and the reasoning goal is stored in a dedicated variable. Reasoning is effected by a form of backward chaining. The following sections detail the reasoning mechanisms implemented by the Control Process.

2.3 Rule-Based Reasoning

To facilitate reasoning, BRAINN's assemblies are labelled with the type of information they store: SN for a (semantic net's) relation, LHS for a rule's left-hand side, and RHS for a rule's right-hand side. The label is represented by a unique sequence of 4 bits, stored in a few additional units in each assembly. Hence, each assembly actually consists of $3N + 4$ units.

As previously stated, BRAINN's rule-based reasoning engine implements a form of backward chaining. The pseudocode for the algorithm is described in Figure 9.

If more than one rule can be used, the rules are sorted by ascending number of conditions in their LHS. The algorithm checks that an LHS condition is satisfied by (recursively) asking the network to produce its value. For example,

```
ApplyRule(question)

1. Deliver question to all assemblies
2. Relax the network
3. If there is a SN assembly containing question Then return
   corresponding answer
4. Else
   (a) For all RHS assemblies containing question
        i. Retrieve LHS of rule
       ii. Sort rules by ascending number of LHS conditions
   (b) For all rules in above order
        i. Load rule to STM (both RHS and LHS assemblies)
       ii. For each LHS condition of rule
             – If LHS.value ≠ ApplyRule(<LHS.object, LHS.attribute, ?>))
               Then try next rule
      iii. Give the answer from RHS of rule
```

Fig. 9. BRAINN's Backward Chaining Algorithm

the algorithm checks the condition sky has_colour blue by asking the question <sky, has_colour, ?>. Although adequate for single-valued attributes, this may cause problems for multi-valued attributes.

The following illustrates the working of the rule application algorithm on a simple reasoning task. Assume that BRAINN's knowledge base consists of the following relation and rule:

<Garfield, drinks, milk>
IF <&someone, drinks, milk> THEN <&someone, is, strong>

For simplicity, also assume that the hexagonal network consists of only 3 assemblies, organised as shown in Figure 10. The divisions in the assemblies represent subsets of units, one for each element of information (i.e., object, attribute, value and label). Also assume that the relation is stored in the upper assembly and the rule in lower and right assemblies as shown in Figure 10.

Fig. 10. Simple Hexagonal Network

The simplest question that the user can ask, is about what Garfield drinks, i.e.,

<Garfield, drinks, ?>

The algorithm delivers the question to all the assemblies. The network, after relaxation, is shown in Figure 11. A label over a division denotes that the activation of units in that division is equal to the binary representation of that label. If there is no label over a division, the network is in a spurious attractor.

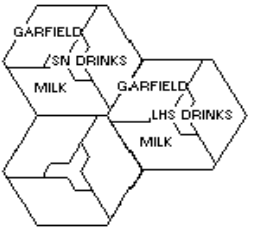

Fig. 11. Network after Relaxation: <Garfield, drinks, ?>

After relaxation, the bottom assembly is in a spurious attractor, the right assembly has settled to one of the patterns stored in that assembly, and the top assembly has settled to the relation (tag is SN) containing the question. Hence, the system gives the answer from this assembly, i.e., milk.

The following question, which asks what Garfield is, causes BRAINN's rule-based reasoning mechanisms to be applied.

<Garfield, is, ?>

As before, the question is delivered to all the assemblies. The network, after relaxation, is shown in Figure 12.

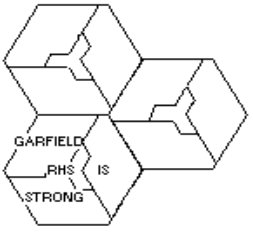

Fig. 12. Network after Relaxation: <Garfield, is, ?>

Two assemblies are empty, because the network has settled in spurious attractors. Sequences of bits in those assemblies have no meaning. The lower assembly

stores the RHS condition, which contains the question. Neighbours of that assembly receive its pattern of activation multiplied by the matrices of weights between assemblies. The resulting network is shown in Figure 13.

Fig. 13. Network after Retrieving LHS of Rule

The upper assembly is clear because the weights between the upper and lower assemblies are equal to zero (no rule is stored). In the right assembly, the LHS has been retrieved. The rule, IF <Garfield, drinks, milk> THEN <Garfield, is, strong>, is written to STM and the question, <Garfield, drinks, ?>, is delivered to the network. The behaviour of the network for this question is as described above. The value returned is the same as the value in the LHS of the retrieved rule, hence the answer for the question, <Garfield, is, ?>, is given from the RHS of the rule, i.e., strong.

Currently, the system cannot answer questions involving variables (e.g., <?, is, strong>) since, after relaxation, the assembly which stores the RHS has one part (i.e., the object) empty or without meaning. Further work is necessary to overcome this limitation.

2.4 Similarity-Based Reasoning

In addition to encoding relations and rules as described in section 2.1, BRAINN learns and reasons from similarity, as shown below. Consider the knowledge database shown in Figure 14.

Fig. 14. Sample Knowledge Base

The database contains some information about cars, planes and lorries. The user may ask "What is a lorry used for travelling on?" The database does not contain the answer explicitly. However, lorries and cars have more attributes in common than lorries and planes. In this sense, a lorry is more similar to a car than to a plane. Hence, the system can guess that a lorry is for travelling on the ground like a car.

To increase the capacity of the network, BRAINN generally stores new information in the assembly where the most similar information is already present. Two mechanisms are then available for similarity-based reasoning, one using on a voting algorithm and the other relying on Pavlov-like connections.

Voting Algorithm The voting algorithm assumes that all the relations with the same object are stored in the same assembly. The algorithm to retrieve the value of attribute Aquery for object Oquery is described in Figure 15.

```
All the relations with object Oquery are retrieved from memory
(one-by-one from the same assembly, using relaxation)
For each retrieved relation <Oquery, Aretrived, Vretrived>

1. Aretrived and Vretrived are delivered to each assembly
2. For each assembly C
     – If ∃Osimilar s.t. C stores a relation <Osimilar, Aretrived,
       Vretrived> Then
         • If ∃Vsimilar s.t. C stores a relation <Osimilar, Aquery,
           Vsimilar> Then vote for Vsimilar

Choose the value with the largest number of votes as the answer
```

Fig. 15. Voting Algorithm

For example, assume BRAINN implements the knowledge database presented in Figure 14. If asked <lorry, is_for_travelling, ?>, BRAINN would assert on_ground since lorry shares two properties with car (made_of metal + has wheels = 2 votes for on_ground) and only one with plane (made_of metal = 1 vote for in_air).

Pavlov-like Connections The algorithm based on Pavlov-like connections assumes that all the relations with the same attribute and the same value are stored in the same assembly, whilst relations with the same attribute but different values are stored in different assemblies. The hexagonal network is overlaid with a fully connected mesh. These additional connections between all the assemblies capture co-occurring features (e.g., if some values of some attributes occur together for one object, then some of the assemblies are active together). The strengths of these Pavlov's connections represent how often assemblies are active together. When a new relation is learnt then:

1. The object from this relation is sent to all the assemblies
2. The assemblies are relaxed (only the assemblies that remember any information about the object are in "resonance")
3. The strength of all the Pavlov's connections between the assembly where the new relation is stored and the assemblies in resonance is increased

Figure 16 shows the Pavlov's connections for the knowledge database of Figure 14. Only non-zero connections are shown. Line thickness is proportional to strength.

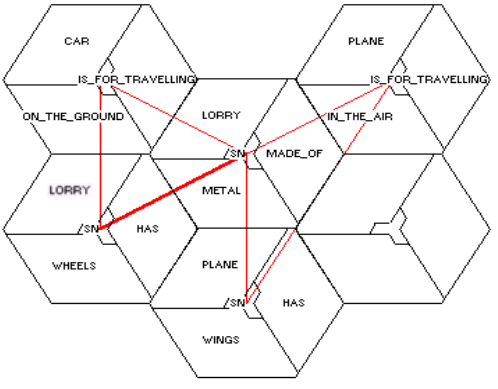

Fig. 16. Pavlov-like Connections

The algorithm to answer the question <Oquery, Aquery, ?> is in Figure 17.

```
The pattern of the object Oquery is sent to all the assemblies and all
the assemblies that remember any information about the object Oquery are
activated
Pavlov's connections are used to determine which value of the attribute
Aquery usually occurs with the set of features of the object Oquery
```

Fig. 17. Pavlov Algorithm

For example, consider the behaviour of the system for the question: <lorry, is_for_travelling, ?>. The object lorry is sent to all the assemblies and two assemblies storing information about the lorry are activated (see Figure 16). The assembly remembering the relation <?, is_for_travelling, on_the_ground> receives activation from two assemblies and the assembly remembering the relation <?, is_for_travelling, in_the_air> from one assembly only, hence the system will guess that the answer is: on_the_ground.

The advantage of the voting algorithm is its simplicity and relatively low computational cost (computations are strongly parallel). The Pavlov-like algorithm is slightly more involved, but has very interesting feature - connections

between features are remembered even if the particular cases are forgotten (also a feature of human learning). Therefore, such connections could be used for rule extraction.

3 Empirical Results

BRAINN is implemented in C++ under Windows, with a GUI displaying traces of the network's behaviour (see http://www.cs.bris.ac.uk/ bogacz/brainn.html). Results of preliminary experiments with BRAINN follow.

3.1 Classical Reasoning Protocols

Several tasks from the set of Benchmark Problems for Formal Nonmonotonic Reasoning [8] were presented to BRAINN. The system incorporates the premises and correctly derives the conclusions for problems A1, A2, A3, A4, B1 and B2, which include default reasoning, linear inheritance and cancellation of inheritance.

3.2 Sample Knowledge Base

BRAINN was also tested with a more realistic knowledge base in the domain of soil science. This knowledge base consists of 20 rules with up to 2 conditions in the LHS (e.g., IF <&soil, is, clay> AND <&soil, humus_level, high> THEN <&soil, compaction, low>) and chains of inference of length 3 at most.

The following is an example of BRAINN's reasoning after "learning" the soil science knowledge base.

```
User inputs: <My_soil, colour, brown>
             <My_soil, weight, heavy>
User query:  <My_soil, compaction, ?>
```

The question is sent to all the assemblies. After relaxation, no SN assembly is found with the answer, but there is a RHS assembly that contains an answer. This RHS assembly belongs to the rule, IF <&soil, Fe_level, high> AND <&soil, weight, heavy> THEN <&soil, compaction, high>. The RHS assembly sends its weight-multiplied pattern to all of its neighbours. After relaxation the two LHS neighbours are found. The rule is retrieved and written to STM. Then, the first condition, <My_soil, Fe_level, high>, is sent to all the assemblies. Again, no SN assembly is found, but the RHS assembly of the rule, IF <&soil, colour, brown> THEN <&soil, Fe_level, high>, is activated. This second rule is retrieved (as above) and written to STM. The condition of the rule, <My_soil, colour, brown>, is then sent to all the assemblies. After relaxation, one of the assemblies still contains the condition so that <My_soil, Fe_level, high> is confirmed. The system sends the second condition, <My_soil, weight, heavy>, of the first rule from STM to all the assemblies. After relaxation one of assemblies contains the condition. Both conditions of the first rule are now confirmed and the system produces the answer, high, from the RHS of the first rule. Although BRAINN behaves as expected, the network is large (36 assemblies of 29 units each).

4 Related Work

The BRAINN system is inspired by some of Hinton's early work [6]. In Hinton's system, as in BRAINN, relations are implemented in a neural network in a distrributed fashion. However, a different network architecture is used.

Many hybrid symbolic connectionist systems have been proposed. One of the first such systems is described in [14]. That system imitates the structure of a production system and is made up of several separate modules (working memory, production rules and facts). With its distributed representation in all of the modules, the system can match variables against data in the working memory module by using a winner-take-all algorithm. The system has complex structures and is computationally costly. Moreover, it is restricted to performing sequential rule-based reasoning.

CONSYDERR [10] is a connectionist model for concept representation and commonsense reasoning. It consists of a two-level architecture that naturally captures the dichotomy between concepts and the features used to describe them. However, it does not address learning (how such a skill could be incorporated is also unclear) and is limited to reasoning from concepts.

CLARION [13], like CONSYDERR, uses two modules of information processing. One module encodes declarative knowledge in a localist network where the nodes that represent a rule's conditions are connected to the node representing that rule's conclusion. The other module encodes procedural knowledge in a layered sub-symbolic network. Given input data, decisions are reached through processing in and interaction between both modules. CLARION also allows rule extraction from the procedural to the declarative knowledge module.

ScNets [4] aim at offering an alternative to knowledge acquisition from experts. Known rules may be pre-encoded and new rules can be learned inductively from examples. The representation lends itself to rule generation but the constructed networks are complex. Finally, ASOCS [9] are dynamic, self-organizing networks that learn incrementally, from both examples and rules. ASOCS are massively parallel networks, but are restricted to binary classification.

A number of other relevant systems are described in [12]. A thorough review of the literature on hybrid symbolic connectionist models is in [11] and a dynamic list of related papers is in [2].

5 Conclusion

This paper presents a hybrid connectionist symbolic system, called BRAINN (Basic Reasoning Applicator Implemented as a Neural Network). In BRAINN, a hexagonal network of Hopfield networks is used to store both relations and rules. Through systematically orchestrated relaxations, BRAINN supports both rule-based and similarity-based reasoning, thus allowing traditional (monotonic) reasoning, as well as several forms of common-sense (non-monotonic) reasoning. Preliminary experiments demonstrate promise.

Future work will focus on developing Pavlov's similarity based algorithm by implementing rules extraction and integrating similarity-based and rule-based

reasoning into one algorithm. In addition, biological plausibility will be improved by incorporating Goal and STM in the hexagonal network and using local rules to control the behaviour of each assembly and determine the global reasoning process.

Acknowledgements

This work is supported in part by an ORS grant held by the first author. The soil science knowledge base was donated by Adam Bogacz.

References

1. Bogacz, R. and Giraud-Carrier, C. (1998). BRAINN: A Connectionist Approach to Symbolic Reasoning. In *Proceedings of the First International ICSC Symposium on Neural Computation (NC'98)*, 907-913.
2. Boz, O. (1997). Bibliography on Integration of Symbolism with Connectionism, and Rule Integration and Extraction in Neural Networks. Online at http://www.lehigh.edu/ ob00/integrated/references-new.html.
3. Calvin, W. (1996). *The Cerebral Code*. MIT Press.
4. Hall, L.O. and Romaniuk, S.G. (1990). A Hybrid Connectionist, Symbolic Learning System. In *Proceedings of the National Conference on Artificial Intelligence (AAAI'90)*, 783-788.
5. Herz, J., Krogh, A. and Palmer, R. (1991). *Introduction to the Theory of Neural Computation*. Addison-Wesley.
6. Hinton, G. (1981). Implementing Semantic networks in Parallel Hardware. In Hinton, G. and Anderson, J. (Eds.), *Parallel Models of Associative Memory*. Lawrence Erlbaum Associates, Inc.
7. Hopfield, J. and Tank, D. (1985). Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, 52:141-152.
8. Lifschitz, V. (1988). Benchmark Problems for Formal Nonmonotonic Reasoning. In *Proceedings of the Second International Workshop on Non-Monotonic Reasoning*, LNCS 346:202-219.
9. Martinez, T.R. (1986). Adaptive Self-Organizing Networks. Ph.D. Thesis (Tech. Rep. CSD 860093), University of California, Los Angeles.
10. Sun, R. (1992). A Connectionist Model for Common Sense Reasoning Incorporating Rules and Similarities. *Knowledge Acquisition*, 4:293-331.
11. Sun, R. (1994). Bibliography on Connectionist Symbolic Integration. In Sun, R. (Ed.), *Computational Architectures Integrating Symbolic and Connectionist Processing*, Kluwer Academic Publishers.
12. Sun, R. and Alexandre, F. (Eds.) (1995). *Working Notes of IJCAI'95 Workshop on Connectionist-Symbolic Integration*.
13. Sun, R. (1997). Learning, Action and Consciousness: A Hybrid Approach Toward Modeling Consciousness. *Neural Networks*, 10(7):1317-1331.
14. Touretzky, D. and Hinton, G. (1985). Symbols Among the Neurons: Details of a Connectionist Inference Architecture. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'85)*, 238-243.