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 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 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 [4]. The human neocortex is divided into minicolumns (∅ 0.03mm), i.e., groups of neurons gathered around dendrite bundles. Minicolumns create a hexagonal network, and are, in turn, organised in hexagonal macrocolumns (∅ 0.5mm). 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.

![Figure 1. Neuron recruitment](image-url)
macrocolumns can spread through the whole network. This process is believed to be particularly 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.

#### 2.1.1. 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 [1]. Figure 2 shows one such network for \( N=4 \).

![Figure 2. Relation encoding in Hopfield network](image)

For simplicity, Hopfield networks encoding relations are referred to as cells, although this has no biological connotation. Given the above mapping of relations to triples, it is possible that an object may have more than one value for a single attribute. There are two ways to handle such situations. One possibility is to store each triple in distinct cells. The other possibility is to store several triples in the same cell and to use different starting points to retrieve each possible value. Section 2.4 discusses the pros and cons of each solution.

The weights of the network are set using either the Hebb rule or the Perceptron rule [6], 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 cell as shown in Figure 2. That is, 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. 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 [6]. 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 \( x_j' \). 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 as follows:

1. Initialise all weights to 0
2. Repeat
   - For each triple to remember
     - Deliver triple’s elements to the network’s units \( x_i \)
     - For each unit \( i \)
       - Compute the activation \( x_i^{t+1} \)
       - If \( x_i^{t+1} \neq x_i \)
         - Then
           - Update weights from each unit \( j \) (\( i \neq j \)):
             \[
             w_{ij} \leftarrow w_{ij} + x_i' x_j'
             \tag{2}
             \]
   - Until no updates are made or time is out

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.

\[
 x_i^{t+1} = \text{sgn} \left( \sum_{j=1}^{3N} w_{ij} x_j' \right)
\]

If it exists, the network stabilises on the stored pattern most similar to the delivered one. Otherwise, the network is said to stabilise on a spurious attractor.

In BRAINN, all of the questions asked by the user are in 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 a cell 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, the network stabilises in a spurious attractor or on another remembered triple. Examples are given in section 2.3.

#### 2.1.2. 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. As with relations, conditions are represented by triples of the form <Object, Attribute, Value> and subsequently stored in
cells of the form described in section 2.1.1. The various cells representing the conditions of a rule can then be connected into a network of cells that encodes the rule. In the network, there are connections between each unit from the LHS cells and each unit from the RHS cell. The weights between cells 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), cells are organised into a large hexagonal network as shown in Figure 3.

![Hexagonal network of cells](image.png)

With the architecture of Figure 3, each rule may have a maximum of 6 conditions in its LHS. Clearly, rules can also be chained since the RHS of one rule may be used as part of the LHS of another.

When reasoning with rules, BRAINN implements backward chaining. Hence, the network retrieves the LHS of a rule upon delivery of its RHS. The following 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 cell and the four conditions from the LHS are stored in adjacent cells. Upon activation of the cell corresponding to the RHS, the network must retrieve the four conditions of the LHS.

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

Note that LHS cells 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 cells retrieved. To avoid confusion during backward chaining when several rules have the same right-hand sides (e.g., \texttt{IF A AND B THEN C AND IF D THEN C})

The right-hand sides are stored in different cells. If only one cell were used, its activation would cause the retrieval of all conditions, thus artificially creating a single conjunction rather than the correct disjunction of conjunctions. By segregating rules that have the same right-hand sides, it is possible to recover the disjunction by considering them in turn during the reasoning process (see section 2.3).

### 2.1.3. 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 cells. 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. With such a set of weights, the pattern for the variable is sent between cells without any modifications nor interactions with the rest of the cell’s information. The weights inside the LHS cells and the RHS cell 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 cell, 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 (currently) not biologically plausible. A high-level view of BRAINN’s overall architecture is shown in Figure 4.

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).
Memory (STM) and the reasoning goal is stored in a dedicated variable. Reasoning is effected by a form of backward chaining, as follows.

Put the reasoning goal in the Reasoning Goal cell
Clear STM
Until the reasoning goal is reached:
Deliver information from STM and Reasoning Goal to LTM
If there are patterns of activation in LTM representing information that:
a) suits the goal, and
b) is somehow connected to information delivered by STM
Then replace the object used last in STM by this information

The following sections detail the reasoning mechanisms implemented by the Control Process.

2.3. Rule-Based Reasoning

To facilitate reasoning, BRAINN’s cells 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 cell. Hence, each cell 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 as follows.

Rule Application (question)
Deliver question to all cells
Relax the network
If there is a SN cell containing question Then
Return the answer from this relation
Else
For all RHS cells containing question
Retrieve the LHS of the rule
Sort the rules by ascending number of LHS conditions
For all rules in the above order
Load rule to STM (both RHS and LHS cells)
For each LHS condition of the rule
If LHS.value ≠ Rule Application(LHS.object,LHS.attribute,?)
Go to next rule
Give the answer from the RHS of the rule

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, the algorithm checks the condition sky has_colour blue by asking the question sky has_colour ?.

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 cells, organised as shown in Figure 5. The divisions in the cells represent subsets of units, one for each element of information (i.e., object, attribute, value and label).
As before, the question is delivered to all the cells. The network, after relaxation, is shown in Figure 7.

![Figure 7. Network after relaxation: “Garfield is ?”](image)

Figure 7. Network after relaxation: “Garfield is ?”

Two cells are empty, because the network has settled in spurious attractors. Sequences of bits in those cells have no meaning. The lower cell stores the RHS condition, which contains the question. That cell’s neighbours receive its pattern of activation multiplied by the matrices of weights between cells. The resulting network is shown in Figure 8.

![Figure 8. Network after retrieving LHS of the rule](image)

Figure 8. Network after retrieving LHS of the rule

The upper cell is clear because the weights between the upper and lower cells are equal to zero (no rule is stored). In the right cell, 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 algorithm gives the answer for the question, Garfield is ?, 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 cell 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. To increase the capacity of the network, BRAINN generally stores new information in the cell where the most similar information is already present. Two mechanisms are then available for similarity-based reasoning, one using 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 cell. The algorithm to retrieve the value of attribute A-query for object O-query is as follows.

1. All the relations with object O-query are retrieved from memory (one-by-one from the same cell, using relaxation)
2. For each retrieved relation <O-query, A-retrived, V-retrived>
   - A-retrived and V-retrived are delivered to each cell
   - For each cell C
     - If ∃O-similar s.t. C stores a relation <O-similar, A-retrived, V-retrived> Then
     - If ∃V-similar s.t. C stores a relation <O-similar, A-query, V-similar> Then Vote for V-similar

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

For example, assume BRAINN knows that car has wheels, car travels on_ground, car made_of metal, lorry has wheels, lorry made_of metal, plane has wings, plane travels in_air, plane made_of metal. If asked lorry travels ?, 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 cell, whilst relations with the same attribute but different values are stored in different cells. The hexagonal network is overlaid with a fully connected mesh. These additional connections between all the cells capture co-occurring features (e.g., if some values of some attributes occur together for one object, then some of the cells are active together). The strengths of these Pavlov’s connections represent how often cells are active together. When a new relation is learnt then:

1. The object from this relation is sent to all the cells
2. The cells are relaxed (only the cells that remember any information about the object are in “resonance”)

The strength of all the Pavlov’s connections between the cell where the new relation is stored and the cells in resonance is increased

Figure 9 shows the Pavlov’s connections for the above example. Only non-zero connections are shown. Line thickness is proportional to strength. The algorithm to retrieve the value of an object’s attribute is as follows.

1. The object’s pattern is sent to all the cells and all the cells that remember any information about the object are activated
2. Pavlov’s connections are used to determine which value of the attribute usually occurs with the set of features of the object
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 two advantages: 1) connections between features are remembered even if the particular cases are forgotten (also a feature of human learning), and 2) connections could be used for rule extraction. In our implementation, both algorithms are available.

3. Empirical Results

BRAINN is implemented in C++ under Windows, with a GUI displaying traces of the network's behaviour (see http://www.ci.pwr.wroc.pl/~bogacz/brainn). 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 [2]. 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 cells. After relaxation, no SN cell is found with the answer, but there is a RHS cell that contains an answer. This RHS cell belongs to the rule, IF &soil Fe_level high AND &soil weight heavy THEN &soil compaction high. The RHS cell 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 cells. Again, no SN cell is found, but the RHS cell 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 cells. After relaxation, one of the cells 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 cells. After relaxation, one of cells 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 cells of 29 units each).

4. Related Work

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 very 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 neural network. Given input data, decisions are reached through processing in and interaction between both modules. CLARION also allows rule extraction from the procedural knowledge module to the declarative knowledge module.

ScNets [5] 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 they 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 and algorithms is in [11], and a dynamic list of related papers is in [3].

5. Conclusion

This paper presents BRAINN, a hybrid connectionist symbolic system. 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 includes:

- Further integrating similarity- and rule-based reasoning (e.g., applying rules to uncertain results derived by similarity, using similarity as a fall-back when no rule applies, etc.).
- Extending the knowledge representation to “non-triple” relations (e.g., giving, which involves giver, given and receiver) and higher-level constructs (e.g., meta-rules, functions).
- Revising the storage policies to overcome the “grandmother-cell” effect and increase capacity.
- Improving biological plausibility by incorporating Goal and STM in the hexagonal network and using local rules to control the behaviour of each cell and determine the global reasoning process.

References


