Connectionist Propositional Logic

A Simple Correlation Matrix Memory Based Reasoning System

D Kustrin and **J Austin**
Advanced Computer Architecture Group
Department of Computer Science
University of York, York. YO10 5DD, UK
(dan|austin)@cs.york.ac.uk

Abstract

A novel purely connectionist implementation of propositional logic is constructed by combining neural Correlation Matrix Memory operations, tensor products and simple control circuits. The implementation is highly modular and expandable and in its present form it not only allows forward rule chaining but also implements *is a* hierarchy traversal which results in interesting behaviour even in its simplest form. Additionally, some biological relevance can be inferred from the architecture given Hebbian learning mechanism, CMM construction and utilisation of inhibitory synapses for control.

Introduction

Correlation Matrix Memories (CMMs) are simple binary feed forward neural networks which have recently been applied to a large number of real world applications in domains as diverse as molecular matching (Turner & Austin 1997b), image segmentation (O'Keefe & Austin 1996) and financial forecasting (Kustrin, Austin, & Sanders 1997). CMMs have been studied by various research establishments over the years from initial theoretical work by Køhonen (1978) to more modern works like Sommer and Palm (1999) and Turner (1997a). Their well understood semantics, complexity and storage properties makes them an ideal vehicle for preliminary research into connectionist reasoning systems since their operation characteristics are known and a number of efficient implementations (both in software and in hardware) exist (Kennedy *et al.* 1995). In this paper CMMs have been employed as execution engines for a propositional logic implementation as well as for implementation of an *is_a* hierarchy. The actual system has been implemented as an interactive interpreter in which only the parser and proposition-to-code translation was not built from connectionist components: all actual processing is performed on a purely connectionist architecture. In its operation it is most similar to Smolenski's system (Smolensky 1990) but in contrast to their architecture CPL is a purely binary connectionist system with defined semantics and integrated *is_a* hierarchy. Additionally it is also highly modular and thus scalable. CPL builds on initial CMM in reasoning theoretical papers (Austin 1994; 1995) and represents a step in evolution towards connectionist higher order logic implementations.

The following section introduces the basic CMM operations, atom binding and coding procedures, Architectures section presents the actual implementation while execution section contains an annotated sample transcript from an interactive session with the system.

Operational Semantics

The CMMs, being equivalent to single layer binary neural networks, have simple teaching and recall algorithms. Their power lies in application of thresholding and their ability to process overlapped input vectors. Teaching of a matrix M given input vector I and output vector O is performed as

$$M' = M \bigvee OI^{\mathrm{T}} \tag{1}$$

and recall is simply

$$O = \Theta_\theta (MI) \tag{2}$$

where Θ_θ is a transfer function performing a threshold at level θ. Thresholding is a very powerful technique which allows extraction of correct associations even in presence of some noise; by manipulating threshold level it is possible to extract not only the matching association but also outputs which are "close" to it. Since CMMs work only on fixed weight binary vectors it is necessary to convert all input/output values into appropriate form. For propositional logic random binary vectors are generated for each term; these vectors are not necessarily orthogonal but are ensured to be sparse (few bits set) which allows accurate and almost error-free recalls. Error properties will not be discussed in this paper, see (Turner & Austin 1997a) and (Sommer & Palm 1999) for further information on CMM error properties. It is interesting to notice that it has been shown that binary CMMs exhibit lower crass-talk then non-binary single layer feed-forward neural networks for same associative tasks, which provides additional validation for use of CMMs (O'Keefe 1997).

The implemented connectionist propositional logic (CPL) is a slight extension of the traditional propositional logic since it recognises two types: propositions and atoms. Each proposition is formed by binding of two atoms and all rules (logic operations) are performed exclusively on propositions. Atoms were introduced to allow for more complex behaviour and are a direct consequence of introduction of a *is_a* hierarchy. Additional level of complexity is introduced by having two modes of operation: learning of axioms and resolution. Before presenting each logic operation and its semantics it is necessary to define CPL syntax.

Sentence	$\rightarrow$	AxiomSentence
Sentence	$\rightarrow$	ResolveSentence
AxiomSentence	$\rightarrow$	**AXIOM** AxiomTail
AxiomTail	$\rightarrow$	IsaTerm
AxiomTail	$\rightarrow$	AxiomExp $\Longrightarrow$ AxiomExp
AxiomExp	$\rightarrow$	BindTerm
AxiomExp	$\rightarrow$	(AxiomExp)
AxiomExp	$\rightarrow$	BindTerm $\wedge$ AxiomExp
ResolveSentence	$\rightarrow$	**RESOLVE** ResolveExp
ResolveExp	$\rightarrow$	(ResolveExp)
ResolveExp	$\rightarrow$	ResTerm
ResolveExp	$\rightarrow$	$\neg$ ResolveExp
ResolveExp	$\rightarrow$	ResTerm $\wedge$ ResolveExp
ResolveExp	$\rightarrow$	(ResolveExp) $\Longrightarrow$ (ResolveExp)
ResTerm	$\rightarrow$	BindTerm
ResTerm	$\rightarrow$	IsaTerm
IsaTerm	$\rightarrow$	Atom $\xrightarrow{is\text{-}a}$ Atom
BindTerm	$\rightarrow$	Atom $\otimes$ Atom

Figure 1: A grammar for well-formed formulae in CPL.

Syntax

The alphabet of the CPL is defined as

$$\Sigma = \left\{\mathcal{A}, \mathcal{P}, \neg, \Longrightarrow, \xrightarrow{is\text{-}a}, (,), \wedge, \otimes\right\} \tag{3}$$

where

$$\mathcal{A} = \{\top, \bot, \mathsf{any}, \mathsf{none}, \mathsf{a}_1, \mathsf{a}_2, \dots\} \tag{4}$$
$$\mathcal{P} = \{\mathsf{p}_1, \mathsf{p}_2, \dots\} \tag{5}$$
$$\mathsf{p}_i = \mathsf{a}_j \otimes \mathsf{a}_k \tag{6}$$

operator $\otimes$ is the binding operator. The truth values, $\top$ denoting truth and $\bot$ denoting false are special atomic constants as are any and none. It is immediately obvious that the disjunction connective is not part of the alphabet and that propositions, p_i, are constructed from atoms by application of the binding operator, $\otimes$. Due to the operational semantic of the CMMs (see below) it is possible to convert disjunctive expressions into CPL well-formed formulae. Any expression of the form:

$$\mathsf{p}_i \vee \mathsf{p}_j \vee \dots \Longrightarrow \mathsf{p}_q \tag{7}$$

can be expressed as

$$\begin{array}{ccc} \mathsf{p}_i & \Longrightarrow & \mathsf{p}_q \\ \mathsf{p}_j & \Longrightarrow & \mathsf{p}_q \\ \vdots & \Longrightarrow & \mathsf{p}_q \end{array} \tag{8}$$

and any expression of the from

$$\mathsf{p}_i \Longrightarrow \mathsf{p}_j \vee \mathsf{p}_k \vee \dots \tag{9}$$

can be expressed as

$$\begin{array}{ccc} \mathsf{p}_i & \Longrightarrow & \mathsf{p}_j \\ \mathsf{p}_i & \Longrightarrow & \mathsf{p}_k \\ \mathsf{p}_i & \Longrightarrow & \vdots \end{array} \tag{10}$$

These conversion forms allow rewriting of any disjunctive expression in CPL *wff*. Given the CPL alphabet it is necessary to present the grammar which shows the difference between atomic and propositional sentences as well as two modes of operation, see fig. 1.

Strictly speaking the grammar is ambiguous since it cannot be presented in LL(1) form. To solve this problem precedence ordering is defined (in the parser) as follows (from highest to lowest): $\otimes, \xrightarrow{is\text{-}a}, \neg, \wedge, \Longrightarrow$.

Semantics

The easiest and most accurate way of describing CPL implementation is defining its semantics, given basic CMM and binary vector operations. The semantics of the CPL is closely tied to CMM operation and their properties. A CMM is a binary feed-forward neural network which accepts and produces fixed length, fixed weight binary vectors. Consequently, each atom has to be converted into such a vector; in the implemented architecture this is done by assigning a random fixed length, fixed weight binary vector to each atom. Additionally, two out of four special atomic propositions (any and none) have special forms: any is represented by a vector with all bits set while none is a zero vector. The binding operator, $\otimes$, performs a outer product (tensor) between two atoms and produces a vector which given atom vector length of n and weight of k will be of length n^2 and weight k^2:

$$\mathsf{a}_i \otimes \mathsf{a}_j = \mathsf{a}_i \otimes \mathsf{a}_j = \begin{bmatrix} b_1^1 & b_1^2 & \cdots & b_1^n \\ b_2^1 & b_2^2 & \cdots & b_2^n \\ \vdots & \vdots & \ddots & \vdots \\ b_n^1 & b_n^2 & \cdots & b_n^n \end{bmatrix} \tag{11}$$

$$= \begin{bmatrix} b_1^1 & b_1^2 & \cdots & b_n^n \end{bmatrix} \tag{12}$$

where $b_l^k = \mathsf{a}_i^k \times \mathsf{a}_j^l$. Outer product is used for binding as it allows superposition of propositions without loss of association.

As stated above, a CMM is a type of a neural network it has two modes of operation: training and recall. These two modes are mapped on to axiom presentation and resolution. The training process can be defined as an operator action ($\mathfrak{T}$) between two sets of binary vectors $\mathbf{I} \in \mathbb{B}^n$ and $\mathbf{O} \in \mathbb{B}^n$, denoting relational mapping $\mathbf{I} \to \mathbf{O}$, and a CMM, M as follows:

$$\begin{aligned} \mathfrak{T}\left(\mathbb{B}^n, \mathbb{B}^n, \mathbb{M}^{n\times n}\right) &\to \mathbb{M}^{n\times n} \\ \mathfrak{T}\left(\mathbf{I}, \mathbf{O}, M^i\right) &= M^i \vee \mathbf{O}\mathbf{I}^{\mathrm{T}} = M^{i+1} \end{aligned} \tag{13}$$

given $M^0 = 0$. This equation shows that training is just a disjunction of outer products. Similarly, recall operator ($\mathfrak{R}$) takes a CMM, input vector and an integer (threshold) and produces an output vector:

$$\begin{aligned} \mathfrak{R}\left(\mathbb{B}^n, \mathbb{M}^{n\times n}, \mathbb{Z}\right) &\to \mathbb{B}^n \\ \mathfrak{R}\left(\mathbf{I}, M, n\right) &= \Theta_n\left(M\mathbf{I}\right) = \mathbf{O} \end{aligned} \tag{14}$$

The threshold parameter can be varied to produce various effects but in this paper it is assumed to be set at the input/output vector weight at all times, see (Austin 1995) for further information on application of thresholds in CMMs.

These three definitions (binding, training and recall) allow specification of the semantics of BindTerm, IsaTerm, AxiomTail implication and ResolveExpression implication terms in the grammar:

$$\text{Atom} \otimes \text{Atom} \quad : \quad \left(\mathbb{B}^n, \mathbb{B}^n\right) \to \mathbb{B}^{n^2} \tag{15}$$

$$\triangleq \quad \mathsf{a}_i \otimes \mathsf{a}_j \tag{16}$$

$$\begin{aligned} \text{Atom} \xrightarrow{is\text{-}a} \text{Atom} \quad &: \quad \left(\mathbb{B}^n, \mathbb{B}^n\right) \\ &\triangleq \quad \mathsf{a}_i \xrightarrow{is\text{-}a} \mathsf{a}_j \\ &\triangleq \quad M_{is\text{-}a} \leftarrow \mathfrak{T}\left(\mathsf{a}_i, \mathsf{a}_j, M_{is\text{-}a}\right) \end{aligned} \tag{17}$$

$$\begin{aligned} \text{Atom} &\xrightarrow{is\text{-}a} \text{Atom} \\ &: \quad \left(\mathbb{B}^n, \mathbb{B}^n\right) \to \mathbb{B}^{n^2} \\ &\triangleq \quad \mathsf{a}_i \xrightarrow{is\text{-}a} \mathsf{a}_j \\ &\triangleq \quad \mathsf{a}_j = \mathfrak{R}\left(\mathsf{a}_i, M_{is\text{-}a}, \theta\right) \end{aligned} \tag{18}$$

where $\left(\mathbb{B}^n, \mathbb{B}^n\right) \to \mathbb{B}^{n^2}$ (for example) is the type definition, first row is the grammar representation and the second row is the implementation definition. The above two forms of the *is-a* association relate to either training (axiom presentation), eq. 17, or recall (resolution), eq. 18. The resolution equation (eq. 18) returns a truth value which is a vector of length n^2 and weight k^2, see below. Similar equations can be constructed for the implication connective:

$$\begin{aligned} \text{BindTerm} &\implies \text{BindTerm} \\ &: \quad \left(\mathbb{B}^{n^2}, \mathbb{B}^{n^2}\right) \\ &\triangleq \quad \mathsf{p}_i \implies \mathsf{p}_j \\ &\triangleq \quad M_{rule} \leftarrow \mathfrak{T}\left(\mathsf{p}_i, \mathsf{p}_j, M_{rule}\right) \end{aligned} \tag{19}$$

$$\begin{aligned} \text{BindTerm} &\implies \text{BindTerm} \\ &: \quad \left(\mathbb{B}^{n^2}, \mathbb{B}^{n^2}\right) \to \mathbb{B}^{n^2} \\ &\triangleq \quad \mathsf{p}_i \implies \mathsf{p}_j \\ &\triangleq \quad \mathsf{p}_j = \mathfrak{R}\left(\mathsf{p}_i, M_{rule}, \theta\right) \end{aligned} \tag{20}$$

As in the eq. 17 and eq. 18, two modes of operation exist: training and recall. In the recall (resolution) the return value is a truth token, see below. The CMM size in *is-a* case and rules case are different. The *is-a* uses matrix of size $n \times n$ and type $\mathbb{M}^{n\times n}$ while rules store uses a CMM of size $n^2 \times n^2$ and type $\mathbb{M}^{n^2 \times n^2}$. The conjunction operator semantics are defined using bitwise 'or' operation ($\bigvee$) as follows:

$$\begin{aligned} \text{BindTerm} &\wedge \text{BindTerm} \\ &: \quad \left(\mathbb{B}^{n^2}, \mathbb{B}^{n^2}\right) \to \mathbb{B}^{n^2} \\ &\triangleq \quad \mathsf{p}_i \wedge \mathsf{p}_j \\ &\triangleq \quad \mathsf{p}_i \bigvee \mathsf{p}_j \end{aligned} \tag{21}$$

As it can be seen from the above equations, the basic return value is a tensored vector of length n^2 and weight k^2. The four primitive atomic propositions $\top$, $\bot$, any and none are, although atomic, of the proposition size and weight since they operate on propositional level. The negation operator can, thus, be defined as

$$\begin{aligned} \neg\text{BindTerm} \quad &: \quad \mathbb{B}^{n^2} \to \mathbb{B}^{n^2} \\ &\triangleq \quad \neg\mathsf{p}_i \\ &\triangleq \quad \begin{cases} \top & \text{if } \mathsf{p}_i = \bot \\ \bot & \text{otherwise} \end{cases} \end{aligned} \tag{22}$$

This definition converts a proposition into a truth value. Although not ideal it is necessary since even under closed world assumption negation semantics is quite difficult. Alternative would be to denote $\neg\mathsf{p}_i$ as the set of all propositions excluding p_i, $\neg\mathsf{p}_i \triangleq \mathsf{all} - \mathsf{p}_i$. This approach was not chosen since it doesn't map well to CMM architecture. The equality operation used in eq. 18 and eq. 20 can be defined as

$$\begin{aligned} \text{BindTerm} &= \text{BindTerm} \\ &: \quad \left(\mathbb{B}^{n^2}, \mathbb{B}^{n^2}\right) \to \mathbb{B}^{n^2} \\ &\triangleq \quad \mathsf{p}_i = \mathsf{p}_j \\ &\triangleq \quad \begin{cases} \top & \text{if } \mathsf{p}_i \text{ and } \mathsf{p}_j \text{ are identical} \\ \bot & \text{otherwise.} \end{cases} \end{aligned} \tag{23}$$

Axioms and Resolution

CPL does not come with any predefined axioms. All axioms have to be presented to the architecture (trained). The resolution system is also very simple. Since both *is-a* and the rules sub-systems are built from the same building blocks they implement the same algorithm: chaining. In *is-a* mode they performs traversal up the *is-a* tree while in the rules sub-system mode they forward chain the rules.

The *is-a* hierarchy is a directed non-cyclical graph depicting both membership and subset relations. In traditional semantic networks there is a distinction between subset relations like $\mathit{Cats} \xrightarrow{Subset} \mathit{Mammals}$ and $\mathit{Bob} \xrightarrow{Member} \mathit{Cats}$ since they necessarily have different semantics, $\mathit{Cats} \subset \mathit{Mammals}$ and $\mathit{Bob} \in \mathit{Cats}$, respectively. In CPL this distinction is not made and a general *is-a* relation is used: $\mathit{Cat} \xrightarrow{is\text{-}a} \mathit{Mammal}$ and $\mathit{Bob} \xrightarrow{is\text{-}a} \mathit{Cat}$. Not making this distinction allows simple application of the binding operator to atoms for formation of propositions. These propositions are then passed to the rule sub-module which uses them for rule searching. Traditionally it has been argued that using *is-a* links instead of separate membership and subset links leads to inconsistencies (McDermott 1976). Although the formal semantics of inheritance has not been developed for this system the problems outlined in (McDermott 1976) are not applicable since the propositions always have the form $\mathsf{class} \otimes \mathsf{query\ atom}$ in which only the class part is modifiable. The class part can only be changed to a superclass until the top of the inheritance graph has been reached. Since the *is-a* hierarchy is assumed to be a directed non-cyclic graph termination is assured.

The rules sub-system accepts propositions and performs forward chaining on left-hand sides of the rules. Given an input rule $\mathsf{p}_i \implies \mathsf{p}_j$ it will search for all the rules matching p_i on the LHS. If none are found it will request modification of the rule LHS by the *is-a* module. This request will ask for replacement of the p_i superclass by its superclass, if any exists, eg. if $\mathsf{p}_i = \mathsf{a}_j \otimes \mathsf{a}_k$ and no rules are matched, the rule sub-system will request the super-class of a_j, say a_m, which will then be bound to a_k into a new proposition $\mathsf{p}_n = \mathsf{a}_m \otimes \mathsf{a}_k$. This new proposition will then be used in another search. The searching process will end when the *is-a* has reached the top of its hierarchy.

Although the architecture only uses forward chaining and a simple semantic network it is capable of performing resolution of a surprising flexibility.

Architecture

The connectionist architecture implementing the semantics of CPL has been built. The grammar and the semantics have been directly implemented in YACC and LEX on a Silicon Graphics Supercomputer. Although currently the architecture is simulated in software it is a truly connectionist artifact. Both *is-a* and the rules sub-systems have been built using exactly the same basic building block presented in fig. 2 (figure is slightly simplified and shows only the recall connections and omits training/recall switching and input/output pre and post processing).

It is interesting to notice that all components used are implementable in purely connectionist form: the Correlation Matrix Memory is a simple binary feed-forward neural network, the Selector components can be constructed by using inhibitory neurons while the comparator is a simple bit-wise conjunction. This necessity for inhibitory synapses for control of recursion is very interesting since it suggests biological plausibility. The control input is used to "stop" the module from forward-chaining and keep it at the current search parameter. This is used, in the complete architecture, by the rules sub-module for control of *is-a* hierarchy traversal. The selector component is used bi-directionally either as a input selector or output selector. Both variants are controlled by a control signal and the component semantics can be defined as follows, in input selection form:

Selector $(I_1, I_2) \rightarrow O$

$$O = \begin{cases} I_1 & \text{if } \mathbf{Control} = 0 \\ I_2 & \text{otherwise} \end{cases} \tag{24}$$

or, in output selection form, as

Selector $(I) \rightarrow O_i = I$

$$i = \begin{cases} 1 & \text{if } \mathbf{Control} = 0 \\ 2 & \text{otherwise} \end{cases} \tag{25}$$

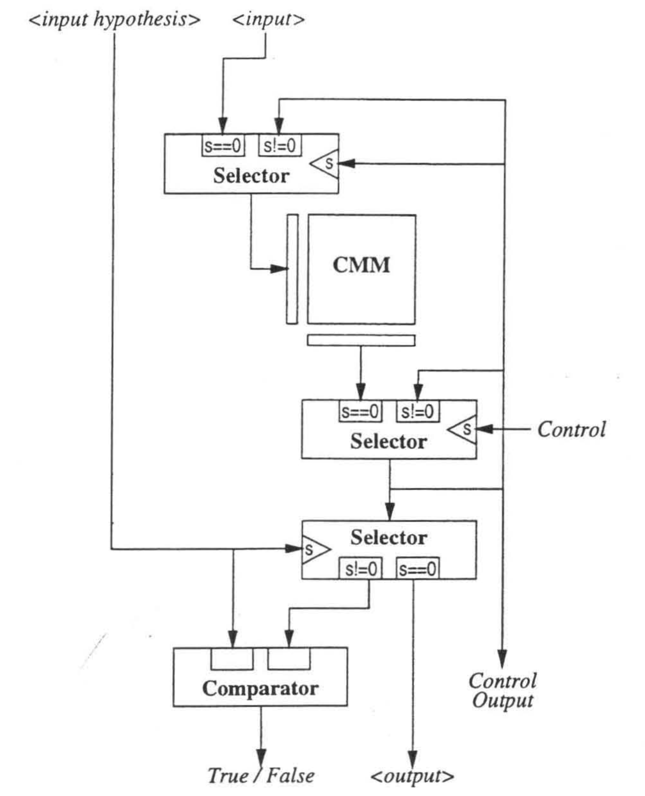

Figure 2: Basic building block for the connectionist architecture implementing CPL. Diagram only shows the data paths for recall mode of operations.

Connectionist implementation of CPL

The complete system is constructed by combining two building blocks via a tensor product ensemble (also implementable as a purely connectionist artifact), see fig. 3.

In the diagram, the left-hand module implements the *is-a* hierarchy while the RHS implements the rule-base. The *is-a* module CMM is of size n while the rules CMM is of size n^2. The system, as depicted in fig. 3 and fig. 2 shows only the resolution connections and control. Training has a different set of connections accessing just CMMs in both modules and is independent of the resolution system.

Execution

The implemented CPL provides an interactive shell for presentation of axioms and for resolution. This section presents an example session which involves learning of a simple semantic network and related rules and some resolution examples.

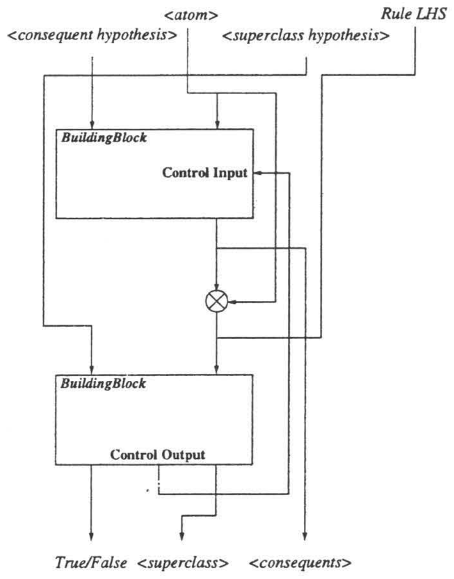

Figure 3: Complete connectionist implementation of CPL. Only recall data paths are shown, as before.

The first set of axioms presented to the systems describe a simple *is-a* graph (fig. 4):

```
AXIOM jim is_a man
AXIOM jim is_a professor
AXIOM dan is_a man
AXIOM dan is_a ra
AXIOM man is_a human
AXIOM professor is_a academic
AXIOM ra is_a academic
AXIOM academic is_a human
```

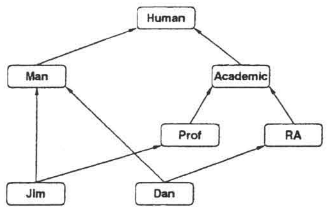

Figure 4: Example *is-a* hierarchy.

while the following rules define how these atoms are interrelated:

```
AXIOM human:any => mortal:true
AXIOM academic:any => mad:true
AXIOM professor:any => has_big_office:true
AXIOM has_big_office:true => important:true
AXIOM man:jim => drinks_tea:true
AXIOM man:dan => drinks_coffee:true
```

Given the set of *is-a* axioms and the rule set it is possible to perform a variety of resolution queries. Resolving an atom searches both the *is-a* set and the rules set:

```
> RESOLVE jim
jim is_a man
jim is_a professor
jim is_a human
jim is_a academic
jim is_a human
jim => drinks_tea:true
jim => has_big_office:true
jim => important:true
jim => mortal:true
jim => mad:true
jim => mortal:true
```

Resolution on a bound atom produces a subset of the results presented above:

```
> RESOLVE man:jim
man:jim => drinks_tea:true
man:jim => mortal:true
```

These examples show the search capabilities. To show the proper rule resolution consider the following examples:

```
> RESOLVE (man:jim) => (has_big_office:true)
No.
> RESOLVE
   (professor:jim) => (has_big_office:true)
Yes.
```

This example shows how the system performs reasonable rule resolution, although jim is both man and professor the property of having a big office is only related to being a professor (via the rule `professor:any => has_big_office:true`). Hence the above example performs correctly. To test rule chaining consider the following result

```
> RESOLVE (professor:jim) =>  (important:true)
Yes.
```

which is the result of chaining rules `(professor:any) => (has_big_office:true)` and `(has_big_office:true) => (important:true)` which suggests that jim is important since he has a big office. Similarly, resolution

```
> RESOLVE (ra:dan) =>  (important:true)
No.
```

shows that dan is not important since he doesn't have a big office.

Since the architecture has no axioms predefined to produce proper *modus ponens* it is necessary to provide the axiom stating that truth implies truth:

```
 AXIOM (true:true) =>
(true:true)
```

which allows for queries like

```
> RESOLVE (((professor:jim =>
              has_big_office:true) AND
             (has_big_office:true =>
              important:true)) =>
             (professor:jim =>
              important:true))
Yes.
```

Conclusion

This paper presented a pure binary connectionist reasoning system based on a propositional logic. The operational semantics were presented as were its modular structure and an example execution. Additionally some interesting parallels with biological systems were highlighted since it is surprising that they should be present in what is largely an engineering approach to building connectionist propositional architecture. This apparent coincidence offers an interesting future research direction.

Acknowledgements

This work is funded by EPSRC ROPA grant GR/L75559.

References

Austin, J. 1994. Correlation matrix memories for knowledge manipulation. In *International Conference on Neural Networks, Fuzzy Logic, and Soft Computing: Iizuka, Japan.*

Austin, J. 1995. Distributed associative memories for high speed symbolic reasoning. *International Journal on Fuzzy Sets and Systems* 82(2):223–233. Invited paper to the special issue on Connectionist and Hybrid Connectionist Systems for Approximate Reasoning.

Kennedy, J. V.; Austin, J.; Pack, R.; and Cass, B. 1995. C-NNAP: A parallel processing architecture for binary neural networks. In *International Conference on Neural Networks (ICANN 95).*

Kohonen, T. 1978. *Associative Memory.* Springer-Verlag, first edition edition.

Kustrin, D.; Austin, J.; and Sanders, A. 1997. Application of correlation memory matrices in high frequency asset allocation. In Niranjan, M., ed., *Fifth International Conference on Artificial Neural Networks.* IEE.

McDermott, D. 1976. Artificial intelligence meets natural stupidity. *SIGART Newsletter* 57.

O'Keefe, S. E. M., and Austin, J. 1996. Document feature recognition using a mesh of associative memories. In *British Machine Vision Conference 1996*, 213–222. BMVA.

O'Keefe, S. 1997. *Neural-Based Content Analysis of Document Images.* Ph.D. Dissertation, Department of Computer Science, University of York.

Smolensky, P. 1990. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence* 46(1-2):159–216.

Sommer, F. T., and Palm, G. 1999. Improved bidirectional retrieval of sparse patterns stored by hebbian learning. *Neural Networks* 12(2):281–297.

Turner, M., and Austin, J. 1997a. Matching performance of binary neural networks. *Neural Networks* 10(9):1637–1648.

Turner, M., and Austin, J. 1997b. A neural network technique for chemical graph matching. In Niranjan, M., ed., *Proceedings of the Fifth International Conference on Artificial Neural Networks.* IEE.