Connectionist Variable Binding

Antony Browne[1] & Ron Sun[2]

[1]School of Computing, Information Systems & Mathematics, London Guildhall University, London EC3N 1JY, UK
abrowne@lgu.ac.uk

[2]Department of Computer Science, University of Alabama, AL 35487, USA
and
NEC Research Institute, Princeton, NJ
rsun@research.nj.nec.com

Abstract: Variable binding has long been a challenge to connectionists. Attempts to perform variable binding using localist and distributed connectionist representations are discussed, and problems inherent in each type of representation are outlined.

Keywords: Neural networks, localist representations, distributed representations, systematicity, variable binding, unification, inference.

1. Introduction

In recent years research on intelligent systems has split into two paradigms, the classical symbolic artificial intelligence (AI) paradigm and the connectionist AI paradigm. Researchers working in symbolic AI maintain that the correct level at which to model intelligent systems (including the human mind) is that of the *symbol*. The symbol is an entity in a computational system that can have arbitrary designations and is used to refer to an entity in the outside world, and the main assumptions on which this paradigm rests were coherently outlined by Newell and Simon (Newell & Simon, 1980) under *the Physical Symbol System Hypothesis* (PSSH). The PSSH states that an entity (or computational system) is intelligent by virtue of the fact that it instantiates a physical symbol system, i.e. a physical system that is engaged in the manipulation of symbols. Symbols are usually taken to correspond to unitary concepts, and can represent objects, events, relations between objects and relations between events. Symbols may be atomic or compound, and at any one time a symbol designates a single entity. Symbols may be combined together to form symbol structures (such as expressions). As an example, the Prolog programming language can contain symbols, such as the atoms **man**, **woman**, **fred** and **mary**. Such symbols can be combined to form more complex symbol structures such as **human(man, fred)** and **human(woman, mary)**. Symbols can also be represented by variables. Such variables can be uninstantiated (i.e. they have not been designated to refer to a specific entity) or instantiated (i.e. they have been designated to refer to a specific entity). For example, the expression **X = fred** designates (binds) the variable **X** to the atom **fred**.

The process of variable binding, where symbols are designated to represent entities, has always been a challenge for connectionist modelers. Many researchers have argued that the ability to perform variable binding is essential if connectionist systems are ever to perform complex reasoning tasks. There are several main points to this argument:

- Variable binding is essential if dynamic structures are to be created in connectionist systems (Feldman & Ballard, 1992; Sun, 1992). As an example, if we have a system that can reason with Prolog structures representing vehicles such as '*blue cars* and *red motorcycles*', the system must have a method of binding the colour *blue* with the vehicle *car* and *red* with the vehicle *motorcycle*, as in**: colour(X,blue), vehicle(X,car)** and **colour(X,red), vehicle(X,motorcycle)**.

- Variable binding is needed to achieve greater computational power in connectionist systems (Sun, 1992), as without variable binding we must have many specific rules rather than one general rule. For example, with variable binding we can have one general rule such as: **dog(X)** $\rightarrow$ **hairy(X), barks(X)**, representing '**X** is a **dog** if **X** is **hairy** and **X barks'**. Without variable binding we must have a specific rule for each eventuality that the system may encounter, such as: **dog(fido)** $\rightarrow$ **hairy(fido), barks(fido) and dog(rex)** $\rightarrow$ **hairy(rex), barks(rex)** and ... etc.

- Variable binding is essential in modeling human cognition. For example, it has been argued that it is necessary for modeling the human language acquisition faculty (Pinker & Prince, 1988) as it imposes constraints on rule matching needed when modeling certain aspects of past-tense learning.

Variable binding can be described as a form of complex pattern matching. There are three main forms of pattern matching used for variable binding, from the simplest to the most complex:

- *Atomic symbol matching* (also called the word problem or template matching), where no explicit variables are present. For example if we have the structure **dog(fido)** stored in a Prolog database and we pose the query **?- dog(fido)**, Prolog answers 'yes'.

- Standard *pattern matching*, where the thing to be matched (the *pattern*) can contain variables, but the thing it is to be matched to (the *datum*) cannot. For example, if **dog(fido)** is stored in the Prolog database and we pose the query **?- dog(X)**, Prolog answers '**X=fido'**.

- *Unification*, where both pattern and datum can contain variables. For example, if the Prolog database contained **dog(Y)** and we pose the query **?- dog(X)**, Prolog answers with something similar to '**X=Y'** (i.e. the uninstantiated variable **X** is bound to another uninstantiated variable **Y**). Unification involving functions can contain something called the *occurs check*, which detects an unsolvable variable binding such as **X=f(X)**. This check is a very costly operation and is omitted in many logic programming systems, however its omission can lead these systems to make incorrect inferences in certain situations. Thus we need to consider it in connectionist systems too.

2. Localist and distributed representations

Researchers working in the connectionist paradigm tend to split into two representational camps, those using *localist* representations and those using *distributed* representations. In connectionist models using localist representations a single unit (or neuron) represents a single concept. In connectionist models using distributed representations, each entity to be represented is represented by a pattern of activity that is distributed over many units. In addition, each unit may be used in representing many different entities. There have been many attempts to define distribution in connectionist systems, the most important ones of which are outlined below.

2.1 Microfeatures

Microfeatures (Hinton *et. al.*, 1986; Hinton 1990) are aspects of the domain that are much 'finer grain' than the primary items of the domain that are to be represented. As an example, to represent a type of

room we associate microfeatures to the things that are usually found in various rooms such as *sofa*, *television*, *ceiling* and *stove*. Each different kind of room can then be represented by a distinctive pattern over these units, the pattern for *lounge* would be different from the pattern for *kitchen*, but would still share some of the microfeatures. Although microfeature-based methods where each microfeature has a semantic interpretation are one way of producing a distributed representation, they are not the only way. It is quite possible to develop a representational scheme where every item is represented by means of a semantically significant feature vector over a set of units without individual microfeatures having any semantic interpretation at all (Collins & Loftus, 1975; Minsky, 1983; van Gelder, 1991).

2.2 Coarse coding

Another way to generate a distributed representation with robustness is by coarse coding (Rosenfeld & Touretzky, 1988). This coding scheme places two conditions on the features assigned to individual units:

1. They must be *coarse*, individual units should be given relatively wide receptive fields (where the receptive field of a unit is defined as those aspects or features of the domain with which the activity of a unit is correlated).

2. The assignments of the units should be *overlapping*, one unit may be involved in representing many different items. The representations of two different items can be superimposed on one another as long as they are orthogonal or near-orthogonal (Smolensky, 1990).

Coarse coded representations are robust, because of the overlapping nature of the receptive fields of the units, no unit is crucial for a successful representation of an item. These representations may also possess semantically significant internal structure, because the particular pattern used to represent an item is determined by the nature of that item. In this way similarities and differences among the items being represented will be reflected by similarities and differences among the representations themselves.

However, coarse coded representations can be constructed where this semantically significant internal structure does not exist (as an example see (Touretzky & Hinton, 1988)). A definition of distribution similar to that of coarse coding, but defined in a different formalism has been put forward by van Gelder (van Gelder, 1991) using the concepts of *extendedness* and *superposition* (c.f. the definitions for coarse coding above). For a representation to be *extended*, the entities being represented by the system must be represented over many units in the system. For a representation to be *superpositional*, individual units in the system must be involved in representing many entities. The representation formed on the hidden layer of a feed-forward neural network can be both extended and superpositional, as the activation patterns of several hidden layer units may respond to a particular input, and each hidden layer unit may respond to several inputs (Sharkey, 1992). A metric has been developed, the *Connectionist Distribution Index* (CDI) (Browne, 1996) which measures the extendedness and superposition present in a feed-forward network by propagating the training set through the trained network and observing how the activation values on the hidden layer change with the presence or absence of particular inputs. A statistical test is then applied to the observed changes in activation value to generate a measure of the extendedness and superposition present in the network, which allows the amount of distribution present in the network to be measured.

2.3 Significance of distributed representation

Those connectionists using distributed representations maintain that the correct level at which to model intelligent systems (including the human mind) lies below the level of the symbol. The main assumptions on which this type of modeling are based were outlined in the '*subsymbolic hypothesis*' (Smolensky, 1988). This hypothesis states that the level of the symbol is too high a level with which to model the mind, and instead of designing programs that perform computations on such symbols, we must design programs that perform computations at a lower level. The correct level is taken to be that of a sub-conceptual connectionist dynamical system that does not lend itself to a complete, formal and precise conceptual level description. This hypothesis states explicitly that a complete description of mental processing at the level of atomic symbols does not exist, and to give a full account of mental processes we must use a description

that lies below the conceptual level of atomic systems. The semantic burden of the system lies at a higher level than that of units or connections, at the level of the distributed representation, a pattern of activity over a number of different units. Because of this distributed representation the system has a complex internal structure that plays an important causal role in the functioning of the system. A distributed representation is not atomic, neither is it a compositional in the symbolic style (i.e., a complex expression composed of constituent atomic representations). Because of this rejection of atomic symbols, supporters of the subsymbolic hypothesis reject the physical symbol system hypothesis. One could argue that, as both neural networks and Von-Neumann type physical symbol systems are both universal Turing machines (Franklin & Garzon, 1990), at one level of abstraction there is no distinction between them. However, the key issue is what constitutes the primitive representation used by these different systems. This is a pertinent issue, both in modeling human cognition and in building intelligent systems.

3. Variable binding with localist representations

Many of the implementations of variable binding attempted so far using connectionist representations have been part of attempts to build inference systems (such as production systems). Two approaches will be described in depth here, binding by sign propagation and binding by phase synchronisation.

3.1 Variable binding by sign propagation

One way of performing variable binding is to use sign propagation (sometimes referred to as signature propagation). The basic idea of sign propagation (Sun, 1989) is very simple: a separate node is allocated for each variable associated with each concept. For example, in first-order predicate logic, each argument of a predicate is allocated a node as its representation; a value is allocated to represent each particular object (i.e., a constant in first-order logic) and thus is a sign of the object which it represents. A node may take on an activation value as a sign in the same way as in conventional connectionist models. This activation value represents a particular object and is merely a pointer. This sign can be propagated from

one node to other nodes when the same object which the sign represents is being bound to other variables from an application of a rule. As an example, consider the rules:

give(X,Y,Z) →own(Y,Z)

buy(X,Y) → own(X,Y)

own(X,Y) → can_sell(X,Y)

A network can be constructed (Figure 1) for representing these rules and for performing reasoning (usually forward-chaining inference is performed in such a network).

Fig. 1. Implementing first-order rules in a connectionist network.

For each predicate in the rule set, an assembly (of nodes) is constructed as a representation of the predicate. The assembly contains *k+1* nodes if the corresponding predicate contains *k* arguments. Each node in the assembly represents one particular argument of the predicate (*argument nodes*), except one node which represents the predicate as a whole (*predicate node*). The argument nodes handle the binding of the corresponding arguments, while the predicate node computes as its activation the confidence (certainty) measure of an entire instantiated predicate. Assemblies are linked up in accordance with rules. If there is a rule containing a condition (e.g., **give(X,Y, Z)**) and a conclusion (e.g., **own(Y,Z)**), corresponding nodes are connected in the assembly representing the condition and in the assembly representing the conclusion. In this process corresponding predicate nodes are always linked up, so that the second predicate node can calculate the confidence for the second assembly based on the confidence of the first assembly, represented by the activation of the first predicate node. For the argument nodes, an argument node in the first assembly is linked to those argument nodes in the second assembly that have the same name as the originating argument node in the rule that is being implemented.

If there are multiple conditions in a rule, the same procedure applies. The predicate nodes of all the assemblies that represent the conditions of the rule are linked to the predicate node in the assembly representing the conclusion. A weighted-sum can be used in the receiving predicate node for evidential combination, i.e. weighing and accumulating evidence received from the conditions in determining the confidence of the conclusion. This process will eventually result in a network as shown in Figure 1.

Forward-chaining inference can be performed with this network. The assemblies that represent known facts are activated, then activations from these assemblies will propagate to other assemblies to which the initial assemblies are connected. Further propagation will occur when those newly activated assemblies pass on their activations to still other assemblies downstream. This process will continue until all the assemblies that are reachable from the initial assemblies are activated. There are two kinds of activations that are propagated: one kind is the confidence measure of a corresponding predicate (which is passed between predicate nodes), and the other is the sign used for representing an individual object (which is passed between argument nodes). These two kinds are completely different, and thus they have separate pathways.

For backward chaining the process is as follows. Firstly a match is attempted between the hypothesis and conclusions of existing rules. If a match is found, then the conditions of the matching rule are used as new hypotheses to be proven. If these new hypotheses can be proven, the original hypothesis is also proven because of the rule. This process is repeated until all of the hypotheses are proven. For example, to prove that **mary** can sell a particular **book**: **can_sell(mary,book1)**, the predicate is matched with the conclusion of the rule: **own(X,Y) → can_sell(X,Y)**. With this match an attempt is made to prove a new hypothesis: **own(mary,book1)**, as if the latter can be proven, the former follows immediately due to the rule above. Assuming that the fact that Mary owns book1 is true: **own(mary, book1)**, this matches the new hypothesis exactly, so the new hypothesis is proven, and consequently the original hypothesis can be proven.

To implement backward chaining with assemblies, in addition to the predicate node another node is needed in an assembly for indicating whether a node is being considered as a hypothesis (the *hypothesis node*). To generate hypotheses backwards, the direction of the link between two hypothesis nodes across two assemblies should be reversed (i.e., the opposite of the direction of the corresponding rule), but the direction of the link between two predicate nodes across two assemblies should remain the same, that is, in the direction of the rule. To start backward chaining inference, the predicate nodes of all the assemblies representing known conditions are activated (but they do not propagate activation). Then the hypothesis node of the assembly representing the hypothesis to be proved is activated. The activated hypothesis node will propagate activation through backward links to generate new hypotheses. If a hypothesis node is activated in an assembly where the predicate node is already activated (in other words, the assembly represents a known fact), then the (backward) activation flow of the hypothesis node is stopped and the activation flow of the predicate node is started (in a forward direction) to activate the predicate nodes of the

assemblies where the activation to the current hypothesis node is from. This forward activation flow continues to activate the predicate nodes of the assemblies on the path from the original hypothesis to the hypothesis that matches the known fact, in the opposite direction of hypothesis generating activation flow. Thus there are two passes involved: one is the backward pass, and the other is the forward pass. The constraints are as follows:

- A hypothesis node can propagate activation to other assemblies in a backward direction, as long as there is no activated predicate node in these other assemblies.

- A predicate node can propagate activation to other assemblies, if and only if these other assemblies have activated hypothesis nodes.

To implement these constraints, some gating mechanism, or the use of conjunctive links, is necessary. In Figure 2, to prove **can_sell(mary, book1)**, we propagate the activation of the hypothesis node backwards to the **own(X,Y)** assembly, using the rule **own(X,Y) → can_sell(X,Y)**. When the hypothesis node of **own(mary,book1)** is activated, it matches exactly the known fact. Therefore, the predicate node of this assembly will propagate activation forward to the predicate node in the assembly of **can_sell(mary,book1)** (which is already instantiated and with its hypothesis node activated), so the original hypothesis is proved.

Fig. 2. A connectionist network for backward-chaining reasoning, where "+" indicates enabling and "-" indicates blocking.

When more complicated rules are represented, such as rules with multiple conditions, the same process is also used. In the case of multiple conditions, when the conclusion of a rule is matched with a hypothesis, multiple new hypotheses corresponding to all of the conditions are generated simultaneously with backward propagation. Later, during forward propagation, the activation from all of these conditions together activates the predicate node of the conclusion.

It is also possible to carry out both forward chaining inference and backward chaining inference simultaneously in one system. In that case, the known facts are activated and the network is run to obtain all plausible conclusions. Rules are treated as specifying certain associations between conditions and conclusions, simultaneously in both directions. Notice that here rules are not implemented in a strict logical sense, in that circular reasoning is not excluded. In this situation, instead of a hypothesis node and a predicate node in an assembly (for representing a predicate), two predicate nodes are used, one forward predicate node (for forward chaining) and one backward predicate node (for backward chaining). Once a forward predicate node in an assembly is activated, the backward predicate node also activated. Once a backward predicate node is activated, the corresponding forward predicate node is also activated. The two

nodes contain the same confidence measure (for the predicate they represent). If both are activated from separate sources, the larger activation value prevails. This situation is illustrated in Figure 2.

One problem with such a network is that some of the activated nodes may quickly rise to saturation, because of mutual bi-directional reinforcement. One way to control activation is to use global inhibition. A global inhibition node measures at each moment the overall activity level of the network (it receives activations from all of the predicate nodes), and then it inhibits all predicate nodes by an equal proportion. That is:

$$a_l(t) = \frac{\sum_j w_j i_j(t)}{a_g(t)}$$

where *l* is any predicate node in the network, *g* is the global inhibition node, and *a* denotes activation. Global inhibition allows the activation of each assembly to reflect their true confidence measure, namely, the evidential support received from input lines.

The Role Binding and Inferencing Network (ROBIN) (Lange & Dyer 1989) and also (Sun, 1989; Sun, 1992) is a connectionist inference system which handles dynamic variable bindings using the mechanism described above. In this model, the terms used are just simple constants and variables. Each constant has a unit in the network that has a uniquely identifiable sign, bindings are formed by passing these signs around the network. A dynamic binding exists when the binding node for a variable has an activation matching the bound constant's sign. These bindings are propagated in parallel and are determined by a constraint satisfaction process, performed using hand coded connection weights. This model only performs simple constant to variable pattern matching rather than full unification. Suggestions have been made that the signs could be distributed patterns of activation that are similar for similar concepts and themselves carry semantic information, but as yet this has not been done.

3.2 A formal treatment of variable binding with sign propagation

Based on the idea explained above, a formal treatment of variable binding with sign propagation will be presented. This treatment will also facilitate possible hardware implementation or software simulation of connectionist rule-based reasoning systems. Note that only forward-chaining inference is dealt with in this treatment, work is currently progressing on applying this formal treatment to backward chaining. In using sign propagation for variable binding, signs have to be propagated and may also need to be manipulated. Such manipulation is quite different from typical connectionist operations, such as weighted-sums. The *Discrete Neuronal Model* (DN model) (Sun, 1989) is a general formalism for connectionist models. It can deal with sign propagation, rule matching and confidence value propagation at the same time. It can be mapped back into conventional connectionist models, such as weighted-sum models. Thus the model forms a unifying framework for complex reasoning in connectionist models incorporating variables.

One question that needs to be answered is exactly what kinds of nodes are needed and what functions should be included in a node in order to deal with sign propagation. An automaton-theoretic description of a node in a DN model as a 2-tuple $W = <N,M>$ is as follows, where:

$N=<S,A,B,I,O,U,V>$

S = the set of all the possible states of a node,

O = the output-line vector,

A = the set of all output values,

I = the input-line vector,

B = the set of all input values,

U = the state transition function,

V = the action function (the output function).

According to the this definition, a DN node has a set of input lines:

$$I = (I_1, I_2, \ldots, I_n)$$

where each element of *I*, I_i, is in *B*, according to the definition of *B*. A DN node has also a set of output lines:

$$O = (O_1, O_2, \ldots, O_m)$$

where each element of *O*, O_i, is in *A*, according to the definition of *A*. Therefore, the value of *I* is in $\prod_n B$, and the value of *O* is in $\prod_m A$ (see Figure 3).

Fig. 3. A DN node.

The state transition and action functions in the previous definition can be explained as follows. The state transition function *U* is actually a mapping, which is from the current state and the current input to the next state; that is:

$$U : S \times \prod_n B \to S$$

The action function *V* is also a mapping, which is from the current state and the current input to the output; that is:

$$V : S \times \prod_n B \to \prod_m A$$

Linking these nodes (with all the requisite mappings) by connecting their input and output lines respectively, creates a network of them. A network of DN nodes can be viewed as a graph with many directed links each of which goes from an output line of a DN node to an input line of another DN node. Due to such linking, there is an additional constraint for this model. If there is a directed link from (the output line of) the node *i* to (the input line of) the node *j*, then $A_i \subset B_j$.

The DN model extends conventional connectionist models to a great extent, and is thus very general. Because of its generality, the DN model can emulate conventional connectionist models. Conversely, a DN node can also be implemented with a network of nodes in conventional models. This formalism should thus be viewed as a descriptive tool and/or a notational convention for presenting and explaining complex network structures.

The DN formalism can be used to construct a network, given some first-order predicate rules. As explained earlier, such a network is made up of assemblies that in turn are made up of DN nodes representing predicates and arguments, that is:

$$Net = def\,(AS, L)$$

where

$$AS = (C, X_1, X_2, \ldots, X_k)$$

k is an arbitrary integer; *C* and *X*'s are a set of DN nodes as specified below, and

$$L \subset (AS_1, AS_2, i, j) \,|\, AS_1, AS_2 \in AS$$

where *i*, *j* are integer labels denoting nodes within an assembly.

The connections among nodes are specified in *L* by labels indicating that a particular node in one assembly is connected to a particular node in another assembly. In each assembly, there are $k + 1$ nodes of the DN type: $C, X_1, X_2,....,X_k$. Here *C* is the predicate node, which contains the confidence value of the predicate (representing the confidence one has on the concept) and is connected to other nodes (mainly other predicate nodes) in other assemblies representing different predicates that are related to it by rules. The argument nodes $X_1..X_k$ take care of variable bindings (for arguments to the predicate) for a total of *k* variables. There may also be connections from each *X* to *C* and from *C* to each *X*. The set of input/output symbols (*A* and *B*) in the argument nodes are values (numeric or symbolic) representing all possible bindings used in a task, the range of which is fixed *a priori* but the meaning can be dynamically assigned. Each node in an assembly has a specific action function, but does not need a state transition function for the present purpose of variable binding. The action functions are as follows:

For the predicate node *C*:

$$O_C^1 = V_C^1(I_1, I_2,..., I_l, O_{X_1}^2,..., O_{X_k}^2) \in A_c$$

$$O_C^2 = V_C^2(I_1, I_2,..., I_l, O_{X_1}^2,..., O_{X_k}^2) \in A_c$$

Note that here O_C^1 and O_C^2 denote the values of the corresponding output lines. Superscript 1 denotes the inter-assembly output and 2 the intra-assembly output. I_i's are inputs received from other nodes (mainly *C* nodes but possibly *X* nodes) in other assemblies impinging on this node (here I_i denotes the value of the input line, not the line per. se.). The *C* node also receives inputs from *X* nodes in the same assembly, so that it can check on their bindings (as will be explicated later). Action function V_C^1 here is a weighted-sum function for evidential combination. V_C^2 produces an *OK* signal to the X nodes to inform them that

everything is checked and is OK so that they can go ahead to propagate bindings (see later examples). Here l is the total number of inter-assembly inputs, so the total number of inputs (intra-assembly and inter-assembly) is $l+k$. A_c is the set of output symbols in the C node. For the argument nodes X_i : i =1, 2,...,k:

$$O^1_{Xi} = V^1_{Xi}(I_1, I_2, ..., I_n, O^2_C) \in A_{Xi}$$

$$O^2_{Xi} = V^2_{Xi}(I_1, I_2, ..., I_n) \in A_{Xi}$$

Note that here O^1_{Xi} and O^2_{Xi} denote the values of the corresponding output lines. Action function V^1_{Xi} here is a mapping specifying the output (the binding) of X_i based on I_i's inputs from X nodes in other assemblies. O^2_{Xi} is the *OK* signal from the C node in the same assembly. V^2_{Xi} produces a signal to the C node to inform the C node about the binding of the argument node (see examples later). A_{Xi} is the set of output symbols in the node X_i. See Figure 4 for a diagram of the interconnections between the predicate node and the argument nodes of an assembly.

Fig. 4. The interconnections in an assembly, where C denotes the predicate node and X denotes argument nodes.

The formalism presented above reveals some subtle timing considerations that need to be addressed. Outputs of a node depend on not only inputs from other assemblies, but also inputs from other nodes in the same assembly, and thus it is important to make sure that there is no mutual waiting (deadlock) situation. In this case, intra-assembly communication from argument nodes X's to the predicate node C (primarily for X's to pass on their bindings so that C can check them) does not depend on inputs from other nodes in the same assembly, only on inputs from other assemblies. So this intra-assembly communication can finish first, and the intra-assembly communication from the predicate node C to

argument nodes *X*'s (the *OK* signal from *C* after checking is done) is facilitated. Then outputs to other assemblies can be computed, without any further impediments. The formalism refines *I* and *O* into two parts respectively, as follows:

$$I = I^1 \circ I^2$$

$$O = O^1 \circ O^2$$

where superscript 1 denotes inter-assembly input/output, superscript 2 denotes intra-assembly input/output, and " $\circ$ " denotes vector concatenation. For action functions:

$$V = V^1 \cup V^2$$

where for argument nodes

$$V^1 : B^I \rightarrow A^{0^1}$$

$$V^2 : B^{I^1} \rightarrow A^{0^2}$$

and for predicate nodes

$$V^1 : B^I \rightarrow A^{0^1}$$

$$V^2 : B^I \rightarrow A^{0^2}$$

In this refined definition, V^2 is done during the first phase of a cycle, communicating within an assembly. V^1 is done during the second phase of a cycle, sending outputs to other assemblies downstream. Note that O^2 (intra-assembly output) of the argument nodes *X*'s is I^2 (intra-assembly input) of the predicate node *C*, and O^2 (intra-assembly output) of the predicate node *C* is I^2 (intra-assembly input) of the argument nodes *X*'s. A simple example is presented for clarification. Suppose there is a rule: 'if something is a table, it is a physical object': **table(X) → phy_obj(X)**. With the rule, when given a fact that **table1** is a table, we can deduce that **table1** is a physical object. Firstly a network is set up for representing the rule. Applying the DN formalism involves build up two assemblies, as shown in Figure 5.

Fig. 5. An example network.

The first assembly represents '**X** is a table' and the second assembly represents '**X** is a physical object'. The input *j* is the sign for **table1**, and the input *i* is the value of the proposition '**table1** is a table', which is one in this case. The output *v* in this case should be the sign for **table1**, and the output *u* should be the value of the proposition '**table1** is a physical object', which should also be one in this case. (states and state transitions in the DN formalism are not needed here.)

Let *R* stand for confidence values, which are real numbers, and ∑ stand for signs, which are also real numbers. Although their domains are the same, they have totally different meanings and are thus represented separately. For *C*1:

$$I = (i, l)$$

$$O = (k, q)$$

where $B_i = R$, $B_l = \sum$, $A_k = R$, and $A_q = R$.

The signals from argument nodes to the predicate node are to inform it whether there is a proper binding present at each argument node. When there is no needed binding, the predicate node should not be activated. The signals from the predicate node to argument nodes are to inform them if they should pass on bindings: when the predicate node is not activated, the argument nodes need not work on binding variables to values. This provides double insurance for practical purposes, although it is not necessary for logical purposes. Now the action function for $C1$ is:

$$V : R \times \sum \ \rightarrow R^2$$

Expressing it in functional forms, we have

$$q = V^1_{C_1}(i,l) = i$$

$$k = V^2_{C_1}(i,l) = 1 \ \ if \ \ i \neq 0, \perp \ otherwise$$

where $\perp$ denotes no output, and q, k, i, and j denote the values of the corresponding input or output lines. The same applies for other input and output lines. For $X1$:

$$I = (j,k)$$

$$O = (l, p)$$

where $B_j = \sum$, $B_k = R$, $A_1 = \sum$, and $A_p = \sum$. The action function is

$$V : \sum \times R \to \sum^2$$

Expressing it in functional forms, we have

$$p = V^1_{X_1}(j,k) = j \ \ if \ \ k = 1, \perp \ otherwise$$

$$l = V^2_{X_1}(j,k) = j$$

It is the same for *C*2 and *X*2. In the case of *C*2:

$$u = V^1_{C_2}(q,r) = q * w$$

$$s = V^2_{C_2}(q,r) = 1 \ \ if \ \ q \neq 0, \perp \ otherwise$$

where *w* is the weight on the link from *C*1 to *C*2 (in this case, we have *w* = 1). For *X*2:

$$v = V^1_{X_2}(p,s) = p \ \ if \ \ s = 1, \perp \ otherwise$$

$$r = V^2_{X_2}(p,s) = p$$

Simplifying the above notations, we can obtain some simple assignment statements, so that they can be easily programmable. Let prime (′) represent the next time period. For *C*1:

$$q' = i$$

$$k' = 1 \ \ if \ \ i \neq 0, \perp \ otherwise$$

For $X1$:

$$l' = j$$

$$p' = j \ \ if \ \ k' = 1, \perp \ otherwise$$

For $C2$:

$$u' = q * w$$

$$s' = 1 \ \ if \ \ q \neq 0, \perp \ otherwise$$

For $X2$:

$$r' = p$$

$$v' = p \ \ if \ \ s' = 1, \perp \ otherwise$$

Two time periods are the necessary duration to reach a conclusion for this simple example. After two time periods, the result can be described as follows:

$$u'' = q' * w$$

and

$$v'' = p' \ \ if \ \ s'' = 1, \perp \ otherwise$$

With the above equations regarding *p*¢, *q*¢and *s*¢, by simple substitution, we get:

$$v'' = j \ \ if \ \ i \neq 0, \perp \ otherwise$$

$$u'' = i * w$$

where *u*¢¢is the output from the predicate node and *v*¢¢is the output from the argument node, all in the '**phy_obj**' assembly. In this case, the input *i* is 1, the input *j* is **table1**, and the weight *w* is 1. So the output *u*¢¢is 1, and the output *v*¢¢is **table1**, which together mean that '**table1** is a physical object'.

One question to be addressed is that, given a limited resolution of activation values in a node, how is it possible to represent all of the domain objects? However, in a reasoning system, only a relatively small set of objects are involved in any particular reasoning task at a given time, and therefore a network with a reasonably fine resolution should be able to handle such a set. A simple calculation suffices: suppose the value range is [-1,1], and the resolution is 0.001, then a total of 2000 objects can be handled at one time. It is possible to dynamically map all relevant objects onto a set of values within the value range of a node. This dynamic mapping (or assignment) can be accomplished by a connectionist network in the form of a simple counter-like circuit, which assigns to each incoming object the (distinct) current value to be used in reasoning, and increments its own value by a small amount for the next object. Another possible method for expanding the available ranges is to use a group of nodes (and a group of links attached to them) for representing the binding of one variable (which in this case has to be represented by a vector), and thus to

achieve much wider ranges than one single node. More nodes can be added to the group to reach any given resolution.

A question arises whether the formalism violates a standard connectionist precept that only simple messages are allowed to pass along links (because the activation values are supposed to represent firing frequencies of 1neurons). The answer is three-fold:

1. A DN node can be mapped to a network of conventional nodes, so that no complex computations and messages are necessary. The DN formalism used here merely provides the convenience necessary for describing complex structures.

2. As indicated by many leading experts, the neural model with ‘activation values representing firing frequencies’ is clearly too simplistic. Although the generic DN may be too powerful to be neurobiologically plausible, its various instantiations used here are more plausible.

3. Similar network models have been advocated and used by some leading researchers, and they have proven advantageous in certain respects. Therefore, it seems justified to adopt this formalism (Aleksander, 1989).

In support of the use of an abstract formalism for dealing with variable binding, notice that what is needed is:

- A rigorous means of description in order to address some complex issues,

- An abstraction that hides away some network details unnecessary for discussing many problem in hand,

- A formal and meticulous model that can reveal subtle problems that might otherwise be overlooked,

- A conceptual tool and a guide for (hardware) implementation (in the future).

In sum, there are enough reasons for using formalisms in general and DN in particular.

CONSYDERR (Sun, 1992; Sun, 1995) is a connectionist architecture based on the DN formalism which integrates rule-based and similarity-based reasoning. It has a two-level architecture, with one level using a localist representation to represent rules, concepts and variable binding, and one using a distributed representation for carrying out similarity based reasoning. Although complex variable binding is possible with this model, amounting to full unification, this system does not perform the occurs check. Whilst the system is based around localist representations, it is suggested how some of the localist nodes could be represented in their own distributed representational space by implementing them using conventional three layer feedforward networks.

3.3 Variable Binding with Phase Synchronization

An alternative way of dealing with variable binding is to utilize the temporal aspects of activation, in addition to or substituting the use of instantaneous activation values. A simple way of utilizing the temporal aspects of activation is phase synchronization. This involves using different phases in an activation cycle to represent different objects involved in reasoning and using in-phase firing to represent binding.

An activation cycle can always be broken into phases. Recall that in the sign propagation method, phases are utilised to make sure that correct values are propagated and no deadlock situation can develop. However, phases can be used in more sophisticated ways to handle variable binding. The activation cycle

can be divided into multiple phases (for example 10), and each of them can be made to stand for a particular domain object (i.e. a
constant) involved in a particular reasoning episode. Each node, representing either an object (constant) or an argument (variable), fires in a particular phase. Each object (constant) node has its own assigned phase throughout a reasoning episode. Each argument (variable) node fires in one of these object phases if it is activated, and is thus bound to the object to which the phase is assigned. See Figure 6 for an example, taken from (Shastri & Ajjanagadde, 1993). In the Figure, **john**, **mary**, and **book1** are objects (constants) and **X**, **Y**, and **Z** are variables, as in **give(X,Y,Z)** (meaning **X** gives **Z** to **Y**). Here, **X** is bound to **john**, **Y** to **mary**, and **Z** to **book1**. Dynamic binding is accomplished by forcing the variables to be bound to fire synchronously, in phase, with the objects (constants) to which they are supposed to be bound.

Fig. 6. Phase synchronization for variable binding.

There are three different types of nodes in a network with this method:

1. Circle nodes, which fire in a particular phase when they are activated in that phase (depicted as small circles in figures).

2. Triangle nodes, which fire in all of the phase of a cycle when they are activated in a phase (depicted as triangles in figures.)

3. Pentagon nodes, which fire in all of the phases of a cycle when they are activated in all of the phases of a previous cycle uninterrupted (depicted as pentagons in figures.)

To perform backward chaining, the following representation can be used. A *k*-ary predicate (with *k* arguments) is represented by a pair of pentagon nodes and *k* circle nodes. One pentagon node is equivalent to the hypothesis node in the previously discussed sign propagation method, and the other to

the predicate node. Each of the *k* circle nodes are the same as the argument nodes before. The hypothesis pentagon node is used in the backward pass, and the predicate pentagon node is used in the forward pass. In accordance with the rules to be implemented, the related hypothesis pentagon nodes (related by a rule) are connected in backward directions (the opposite direction of a rule) and the related predicate pentagon nodes in forward directions (the direction of a rule). Corresponding circle (argument) nodes of the predicates involved are connected in a backward direction (the opposite direction of a rule) as in Figure 7. The same example used when discussing signs for variable binding is presented:

give(X,Y,Z) → own(Y,Z)

buy(X,Y) → own(X,Y)

own(X,Y) → can_sell(X,Y)

The known fact is **give(john,mary,book1)**. When the hypothesis nodes are activated, as dictated by a query, the activation flows backwards to activate those hypothesis nodes that represent the conditions of a rule that has the original hypothesis as its conclusion. These newly activated hypothesis nodes represent new hypotheses, which, once proved, can be used to prove the original hypothesis. This backward flow can continue until they match stored facts.

The facts are wired into the network in the following way. Each circle node for an argument of a predicate that represents a known fact (with all its arguments bound to some constants) is linked conjunctively with, or gated by, the node representing the object (constant) to which the circle node (representing the argument) is bound. If they fire in the same phase, the conjunctive link will pass on activation. If all of such links pass on activations (which means they all match the constants in the known fact), these activations together will activate the predicate pentagon node of the assembly, indicating the hypothesis which the assembly represents is proven. Then the activated predicate pentagon node will propagate activation, in forward directions, to other predicate pentagon nodes in other assemblies. In Figure 7, the original hypothesis is **can_sell(mary,book1)**. The hypotheses **own(mary,book1)** and

give(john,mary,book1) are subsequently generated, one after another. After **give(john,mary,bookl)** is proved with the gating method above, we can further prove those hypotheses that generate this hypothesis, namely **own(mary,book1)** and **can_sell(mary,book1)**, with the forward flow of activation between respective predicate pentagon nodes.

Fig. 7. A network utilizing phase synchronization for variable binding.

The triangle nodes are used to impose constraints that are implicit in the logic rules to be implemented. For example, in implementing **p(X) → q(X,a)**, the **q** assembly needs to enforce the constraint that its second argument must be **a**. This constraint can be accomplished by a conjunctive link that will pass on activation if and only if both the circle node representing the argument and the circle node representing the constant a fire in the same phase. Once they fire in-phase, the triangle node will be activated throughout the cycle, to allow activations from the two circle nodes and the hypothesis pentagon node to pass through as usual since the output from each circle node, as well as the hypothesis node, is gated by the triangle node (see Figure 7). Another example is implementing the equality constraint of two arguments of a predicate as **in p(X) → q(X,X)**. In this case, the constraint can be accomplished by a conjunctive link from both circle nodes representing the two arguments in **q** to a triangle node. The triangle node will be activated, if and only if both circle nodes are activated in the same phase, that is, if and only if they are bound to the same objects. Once the triangle node is activated, it will fire throughout a cycle, and thus enable the two circle nodes and the hypothesis pentagon node to propagate their activations, since the outputs from these nodes are gated by the triangle node (see Figure 7).

In sum, in this model, reasoning is the propagation of temporal patterns (spikes) of activation. Each phase in the pattern correspond to an object; binding is accomplished through synchronous firing of object nodes and argument nodes. Rules are, as before, connection patterns that direct the propagation of activation, while known facts are implemented as subnetworks that gate the activation flow.

One shortcoming of this method is that the initially known facts need to be wired into a network in an elaborate way. Such wiring makes it difficult to reason in a real-time environment, in which situations change constantly and instant attention to these changes and spontaneous reasoning based on these changes are required. Re-wiring will be too slow to account for such processes. Re-wiring is also needed in such simple and frequent tasks as counterfactual reasoning.

A question regarding this method that naturally arises is: do we really need rhythmic patterns in order to accomplish the same computation that is being carried out by the network with phase synchronization? In other words, can we accomplish the same computation, with basically the same network architecture, without involving temporal firing patterns? The answer to the question above is yes. Since temporal patterns add complexity to node computation, it is wise to avoid them if they are not absolutely necessary. One way to accomplish the same task without temporal firing patterns is to utilize the sign propagation method to simulate the phase synchronization method. Instead of generating a temporal pattern within a specific phase, we can generate a sign (i.e. a distinct activation value) that corresponds to a particular phase in the phase synchronization method, so that the binding is accomplished by the matching of sign values instead of by the matching of firing phases (in-phase firing). Although the basic architecture can be preserved, some network details need adjustments. Now that each node that represents a constant will propagate a sign value (instead of firing in a specific phase), the circuit at an argument node will allow the node to accept and pass on the sign value that is passed to it (instead of firing patterns in a specific phase). Gating can be accomplished by comparing sign values (instead of comparing firing phases). Whole-cycle firing is taken care of by a special sign value that can be viewed as potentially the same as any other sign value (especially for comparisons in gating circuits). This solution accomplishes exactly the same task with simpler node computation.

Phase has been used information to bind variables (Ajjanagadde & Shastri, 1989; Shastri & Ajjanagadde, 1993) in a connectionist system *SHRUTI*, where a restricted number of rules with multi-place predicates can be expressed. Queries to the system in which the variable bindings are requested are limited in that

they can contain at maximum of 10 terms (constants or variables), as the authors argue that a biological neuron is unlikely to be sensitive to more than 10 phases. The system can only cope with multiple occurrences of the same variable if this variable is instantiated by the initial query. This implies that the system cannot unify terms such as **f(X,X)** and **f(Y,Z)** where a propagation of the variable binding is required. It is not easy to see how this system can be extended to cope with *n*-ary function symbols where *n* is greater than zero (i.e. function embeddings). However, an extension of this system (Park *et. al.*,1995) has been constructed. This extension contains many structural modifications, allowing it to perform a broader range of inference than *SHRUTI*. In addition to having extended inference capabilities, this network can perform full unification.

3.4 Other localist models

Variable binding was performed by Ballard and Hayes (Ballard & Hayes, 1986) in an attempt to implement a restricted form of the logical inference rule *resolution* using a parallel relaxation algorithm. This system uses simple terms consisting of only unitary constants or variables (no compound terms are allowed). All possible variable substitutions are prewired using hand-coded weights, with incompatible variable bindings being represented by units connected with inhibitory links. Although it has been suggested that the units could be coarse coded this has not been done in practice, possibly because it would make the hand coding of the weights more difficult (for a model performing resolution on distributed representations see (Browne, 1998a).

Frameville (Ananadan *et. al.*, 1989; Mjolness *et. al.*, 1989) is a connectionist frame based system supporting the binding of variables and the dynamic instantiation of frames. This system distinguishes between a static set of hierarchically ordered concepts called the model side (which may contain variables), and a dynamic set of formulae called the data side (which are ground and may not contain variables), both of which can be regarded as directed acyclic graphs with hand coded weights. An objective function is composed representing the mismatch between these two directed acyclic graphs, and

an analog neural network attempts to minimise this function to perform variable binding. As the model only matches variables with constants it is only performing pattern matching, and it seems impossible to extend the model to perform the variable to variable binding required for full unification.

Conposit (Barnden, 1989; Barnden 1994) is a system where data are represented in a grid-shaped network, and production rules are hard-wired into the network's structure. Full unification is not implemented, only a limited form of pattern matching where a rule attempts to find matching data. In addition the variable binding performed by this system can be reduced to a sign propagation mechanism, in much the same way as the phase synchronisation method described above.

Variable binding has been achieved using frequency modulation (Tomabechi & Kitano, 1989) by the propagation of frequency modulated pulses generated by groups of neurons. In effect this is a sign passing model, i.e. it allows some tokens (typically the sources of the activations) to be passed with the activation propagation in the network. Variable binding can be achieved by this ability to specify where a particular oscillation originated from. Only pattern matching is implemented by this model, full unification is not implemented.

Work has been performed using feature structures (Stolke, 1989) to represent terms in unification-based grammar formalisms. Here feature structures are represented as directed acyclic graphs, a valid equivalence relation is defined on the nodes of this graph and this is computed in parallel using a connectionist network This network will perform unification but will not perform the occurs check.

Net-Clause Language (NCL) (Markov, 1991) is a neural network modeling tool which integrates some connectionist and some classical symbolic processing features in a unified computational environment. Although it achieves unification, it cannot be considered to be a true connectionist system as the processing elements do not perform simple numeric calculations, rather they implement complex symbolic algorithms. In addition the network connections, rather than propagating activation values, propagate

complex data structures. This system is best seen as a symbolic system which uses some connectionist ideas to organise its computation in a more flexible way.

A successful implementation of unification has been performed using a localist connectionist network (Holldobler, 1990; Kurfess, 1991). In this system terms to be unified are symbols composed from a finite set, and are either constants (such as **f** and **a**), variables (such as **X** and **Y**) or a *n*-ary function symbol applied to the terms (such as **f(a,X)**, a constant being a 0-ary function). The occurs check is performed by a separate layer of units, the occurs check layer. This is activated after the network has performed its unification, the presence of an occurs check problem in the final term matrix causes a stable mutually excitatory coalition to form between some of the units in the occurs check layer. This stable coalition between the units is then detected, signifying unification failure. This approach to connectionist unification is elegant and effective, generating
the most general unifier of two terms in a number of steps directly proportional to the length of the two terms being considered, and performing the occurs check in two steps for any length of term.

A network that uses quantum mechanical principles *QNET* (Looke, 1995) attempts to perform variable binding using a localist representation where individual units in the network are characterised by amplitudes in more than one frequency. Although this network is an interesting approach, it only achieves pattern matching, not full unification.

4. Variable binding with distributed representations

There have been many different approaches to performing variable binding with distributed representations, so many that it is difficult to categorise the approaches into a few main methodologies as can be done with localist representations. However, major attempts are outlined below.

The Distributed Connectionist Production System (DCPS) (Touretzky & Hinton, 1988) is a connectionist rule interpreter for a restricted class of production systems, where symbolic expressions are manipulated by rules of the form 'if **A** then **B**' where **A** is a condition and **B** is an action. The condition side specifies what expressions must be in the working memory of DCPS in order for the rule to fire, and the action side specifies which expressions should be moved in (or out) of a connectionist working memory when the rule does not fire. A finite set of triples such as **gab** form the symbolic expressions used by the system, and these triples are represented in a distributed fashion in working memory using coarse coding. Part of the system called 'bind space' is used to implement variable binding in this system, and in this system only one variable (**X**) exists. Rules can include this variable, such as the following rule: **Xab, Xcd → +gXp -XrX** (i.e., if the two terms on the left are present add the first term on the right and delete the second). The variable binding performed is very limited as it applies to a very special type of terms, namely those with a variable in the first argument position on the condition side of the rule. No suggestions have been made as to how the system may be modified to cope with variables appearing in other positions.

Tensor product representations have been used to implement variable binding. Connectionists have been aware for a long time that the operation of vector addition can be used to combine two representations to generate a representation of a more complex object, and Smolensky's (Smolensky, 1990) insight was to realise the extra expressive power inherent in a tensor product representation. The tensor product of an *n*-place vector *v* and an *m*-place vector *w* is the *n* m* place vector whose values consist of the pairwise products of the values of the elements of *v* and *w*. In Smolensky's framework, complex objects (such as '**cup of coffee**') are described as being made up of a set of roles which represent argument positions (such as '**contains**' and '**contained by**'), with each role having a certain filler which acts like the variable or constant in that argument position (such as '**coffee**' and '**cup**'). Each role and filler are assigned a distinct vector, fillers are bound to roles by forming the tensor product of their respective vectors. The resulting tensors are added together to form a representation of the whole item. The filler for any role can be extracted from the whole structure by multiplying the tensor representing the whole item by the vector

corresponding to that role. Pattern matching has been carried out using these tensor based representations, but not full unification.

A novel form of connectionist representation, *Plate's Holographic Reduced Representations* (HRR's) has been used to perform variable binding (Plate, 1991; Plate, 1994). Here circular convolution (represented by ⊗) is used to bind variables, for example a term **t** containing the variable **X** bound to the constant **a** and the variable **Y** bound to the constant **b** would be represented by:

$$t = (X \otimes a) + (Y \otimes b)$$

However, this system does not deal with the binding of variables to variables or perform the occurs check, so full unification has not been performed. However unification has been performed by Weber (Weber, 1992) using a variation of HRR's. In Weber's model circular convolution is used to produce distributed vectorial representations of terms, and the unification is then performed on these representations using a feed-forward network. Another connectionist model, the *Self Organising Feature Map* (Kohonen, 1988) has also been used by Weber to perform unification (Weber, 1993) using a localist representation. Both of Weber's models also perform the occurs check.

Full unification has been performed by a neural network using a distributed representational scheme generated by autoassociative networks. An early system performed unification without the occurs check (Browne & Pilkington, 1994), and a later system included the occurs check (Browne & Pilkington, 1995). In the system described here a four-layer connectionist network using sigmoidal activation functions was trained by gradient descent to perform unification on the distributed representations of logical terms. It did this by mapping the distributed representations of pre-unification terms produced by an input autoassociator to the distributed representations of unification results produced by a target autoassociator. The symbolic input representation took inspiration from Holldobler's representation for terms (Holldobler,

1990), but this symbolic representation was then transformed into a distributed representation which the unification was then performed on. The symbolic representations presented to the input autoassociator consisted of 9216 pairs of logical terms, built from two constants (**a** and **f**, which were also used as functors) and two variables (**X** and **Y**). There were four argument positions in the input term pairs, the 0 (functor), 1 (first), 1.1 (nested inside the first argument position) and 2 (second) argument positions (see Figure 8). This gave 32 input units (four variables and constants/functors × four arguments positions × two terms). However, four of these units were redundant, as the variables **X** and **Y** cannot appear in the functor position, in either of the terms. This is illustrated in Figure 8, where the units representing **X** and **Y** at the functor position are represented by broken circles.

Fig. 8. Example symbolic level representation of term pairs presented to input autoassociator.

The remaining 28 input units allowed term pairs such as **f(a(X),Y)** and **f(Y,Y)** to be presented to the input autoassociator in a symbolic form. A similar representational scheme was used for the inputs of the target autoassociator (see Figure 9), with the addition of an extra argument position (the 2.1 position) to allow the propagation of nested arguments across argument positions. For example, the target for the correct unification of the above term pair would be **f(a(X),a(X))**, where the structure **a(X)** has been bound to the variable **Y** and propagated to the second argument position. The target autoassociator had 20 units for representing the unification result (four variables and constants/functors × five arguments positions × one term). Two of these units were redundant, as the variables **X** and **Y** cannot be the functor of the term resulting from the unification. In addition to these 18 units another unit was allocated to signify the results of the occurs check.

Fig. 9. Example symbolic level representation of unification results presented to output autoassociator.

In both the input and target representations a value of 1 was used to signify the presence of a particular variable or constant/functor at that argument position and -1 to signify its absence. The number of units in

the layer of both the input and output autoassociators were chosen using *the Connectionist Distribution Index* (CDI) (Browne, 1996), a metric that measures the level of distribution of input layer representations on the hidden layer. Hidden layer sizes were chosen that both maximised the CDI and allowed 100% reconstruction of the input patterns on the outputs of the respective autoassociators. This gave 21 hidden units for the input autoassociator and 17 hidden units on the output autoassociator. A four layer network (Figure 10) was used for mapping the distributed
representation of the pre-unification terms produced by the input autoassociator to the distributed representation of the unified terms produced by the target autoassociator.

Fig. 10. Four layer network maps distributed representation of pre-unification term pairs from input autoassociator to distributed representation of unification results on target autoassociator.

Initial experiments indicated that for satisfactory unification performance the hidden layer of a three-layer network needed to be larger than that of the input layer. This could be open to the criticism that the distributed representations applied to the input layer were merely being re-expanded to a symbolic representation on the hidden layer. Although this re-expansion to the symbolic level was deemed unlikely, the use of two hidden layers (both of which were not larger than the hidden layer of the input autoassociator) allowed the model to escape this potential criticism. Each of the hidden layers of the network performing unification had 21 units. The 9216 possible term pairs were randomly partitioned into two sets, one set consisting of 80% of all possible terms to be used for training the network, the other 20% to be used for testing the network when training finished. All of the networks involved (both autoassociators and the network performing the unification) were trained using on-line backpropagation (Rumelhart *et. al.*, 1986).

The network successfully unified 97% of the novel term pairs that it had not been exposed to during training. This, to a large extent, demonstrates that unification can be carried out on a distributed representation, as the probability of the network correctly unifying a single term pair by chance is 1 in 2^{21}

(as there are 21 binary output units). The terms that the network failed to unify were found to be different on different runs of the network (using different starting weights and weight ranges). This indicated that the network was not having difficulty in unifying particular types of terms, and the errors were caused by deficiencies in the gradient descent process used for training. Once unification has been performed, an extension of this model was then able to perform the inference procedure of *resolution* on the distributed representations of the unification results (Browne, 1998a).

5. Comparisons

Comparisons can be made between the different representational schemes. Localist representations are popular with many researchers in this field, partially because they are easy to interpret (as each unit in the network is semantically interpretable). It has been argued that as their representations are localised they are in fact symbolic systems, no different from the systems of classical symbolic AI. This is rather a strong position to take, since localist connectionist systems do offer something new as their processing is typically carried out by the massively parallel flow of activation values between units, something that is not commonly seen in classical AI. In the localist networks described in above, no actual variable binding algorithm (such as the unification algorithm) is used, and yet these models perform the variable binding, in ways that are distinct from those performed by classical AI systems.

Distributed connectionist models offer a representational scheme that lies below that of the symbol and are distinguished from classical symbolic AI (or localist connectionist) systems by the way in which their computational (syntactic) features relate to their representational (semantic features). Classical symbolists believe that, in natural language processing, the mechanisms responsible for higher cognitive tasks are supposed to divide into those responsible for syntactically oriented tasks (such as parsing) and those which carry out semantically oriented tasks (such as disambiguation). In connectionist systems using distributed representations, often all the ability to perform the appropriate operations is contained in one set of connection weights, and assignments relevant to both kinds of operation are made in one step. In any

computational system, be it a symbolic AI system or a neural network, the fundamental *syntactic entities* are atomic entities that are manipulated in various ways as the computation proceeds. In a neural network these elementary entities are units (or neurons) and the connections between them, in a symbolic AI system they may be Prolog atoms or their equivalent. On the other hand, in any computational system the fundamental *semantic entities* are the *representational* objects that carry the semantic content of the system. In a distributed connectionist system a representation corresponds to a distributed pattern of activation over many neurons, typically generated by a learning process such as backpropagation (Rumelhart et. al., 1986). In a symbolic AI system a representation might be a Prolog atom or a structure built out of such atoms, whereas in a localist connectionist system representations may be learnt using a technique such as recruitment learning (Diederich, 1988; Diederich, 1991). In symbolic and localist systems the semantic and syntactic representations coincide, whereas in distributed connectionist systems they are separate. As described in the introduction to this paper, often connectionist systems are termed subsymbolic, where the level of computation lies below the level of semantic representation.

In a classical symbolic system a computational token is a featureless chunk given an arbitrary label that distinguishes it from other tokens, e.g. nothing intrinsic to the **dog** token makes it any more closely related to the concept of '*dog*' than to a concept of '*tree*'. There is nothing about the tokens themselves that indicates their meaning. A symbol may be used to designate any expression whatsoever. Formal symbolic tokens such as **dog** get their designated meaning only because of how the system manipulates them and how they interact with the outside world. That is, the arbitrariness of symbol denotations implies that there is no intrinsic meaning attached to a symbol. A representation in a distributed connectionist system is, in contrast, a distributed pattern of activity with a *microsemantics* (Dyer, 1990), an internal pattern that systematically reflects the meaning of the representation. In contrast to a symbolic representation, where any token will serve as well as any other, the internal structure of a connectionist representation is important in the context of an existing system. If we take the distributed representations of two concepts, such as '*dog*' and '*tree*', and make an attempt to exchange them, the system will not function properly. In addition, the internal representation of the concept '*dog*' will in some way be closer

to the internal representation of a concept such as '*cat*', and in some way more removed from a concept such as '*tree*'. In this way connectionist representations carry their own meaning, they are not given meaning in the way that symbolic representations are given meaning, by having it attached to them in an arbitrary fashion.

Because of the way that connectionist representations of similar concepts have similar patterns of activation, connectionist systems possess the ability to generalise and perform correctly on encountering novel input. A novel example will be systematically given an appropriate representation, and the internal structure of this representation will be relatively similar to the structure of representations of objects to which this example is semantically similar in relevant aspects, and quite different from the structure of semantically different objects. Symbolic systems often generalise poorly and sometimes fail completely on encountering novel input, because the symbols in such a system do not carry their own meaning (Sun, 1995b). These aspects constitute significant advantages of distributed representations over symbolic or localist representations.

However, one significant problem of distributed representations is that they can be difficult to interpret. Because of its superposed and extended nature, the distributed pattern of activation representing a concept can not easily be attached to a label, unlike an atom in a symbolic AI system or the activation value of a node in a localist connectionist system. Techniques do exist which attempt to interpret distributed patterns of activation, including statistical techniques (Bullinaria, 1997), rule extraction techniques (Andrews, 1997) and generalised effect analysis (Browne, 1998b), but they are obviously more complex than reading a symbolic representation or recording the activation of a node in a localist connectionist representation. In addition, training times in distributed connectionist systems can be extensive, and it can often be difficult to achieve 100% performance on the designated task.

6. Problems with connectionist representations

There are many problems with connectionist systems (Browne, 1997), but the two major problems with connectionist systems when variable binding is performed, which are not suffered by symbolic AI systems, are described below.

6.1 Limits on the productivity and width of representations

The *productivity* of a system refers to the ability of that system to generate and correctly process items from an infinite set. With reference to the variable binding task, this refers to the processing of an infinite number of variables and constants or functors, in terms of up to infinite length. Because of the possibility of nesting of arguments, computational structures such as terms with other terms nested inside them can be produced. The size of these structures cannot be pre-determined before the run-time of a system. The property of productivity is displayed by symbolic systems which can produce and process an (almost) infinite set of terms from an (almost) infinite set of symbols using recursive structures with an (almost) infinite level of embedding (only being limited by physical constraints such as memory size). Most connectionist systems have a limited set of inputs and can only cope with a limited level of recursion (i.e. nesting of term structures). Because of this, the set of input patterns (variables, constants and terms) that can be correctly generated and processed often need to be completely specified when the network is constructed. Recursive connectionist architectures using distributed representations, such as *RAAM*s (Pollack, 1990) and *XRAAM*s (Lee *et. al.*, 1990) exist, however the width and depth of embedding of the structures represented within them are limited by the precision of implementation, and they cannot produce infinitely larger or deeper structures when required. HRRs (Plate, 1991; Plate, 1994) can be modified in this way, but is unclear whether they can match the full capabilities of a symbolic system in this respect. As an example of this limitation applied to the network performing unification on distributed representations described above (Browne & Pilkington, 1995), the network is constructed to perform unification on terms such as **f(a(X),Y)** but can not process terms containing a novel variable or cope with terms containing a deeper level of embedding such as **f(a(a(X)),Y)**. In a localist connectionist system, new

units would have to be recruited ‘on the fly’ to represent new structures, and the appropriate weightings allocated between the units. How this is to be done remains unclear.

Whether this is a serious limitation on connectionist systems depends on the task they are being used for. If the task is to emulate the properties of a symbolic theorem prover, this is a serious limitation. However, if the task is that of modeling human cognition this may not be such a serious limitation. There is evidence that humans exhibit a limit on the depth of recursion they can process (Dyer, 1995). This is indicated by the difficulty that humans have in understanding sentences containing more than a certain level of embedding. Other evidence relates to the number of elements in a representation that humans can successfully deal with, where the number of elements in a representation affects the level of relation that the representation can represent. The number of elements can be thought of as the number of facets of a situation that can be viewed simultaneously. There is considerable evidence (Halford, 1993) that most humans can reason using representations with at most five elements, such as a predicate (the first element) with four arguments (the four other elements). An example of this would be a predicate ‘**proportion(A,B,C,D)**’, which expresses a relation between four variables such that **A/B = C/D**. In this predicate it is possible for a human to predict how any variable will change in relation to one or more of the others. The psychological existence of the processing of rank six representations (such as a predicate with five arguments) by humans is speculative, and if it exists it probably does so only for a minority of adults. This implies that insofar as matching human cognitive capacity is the of intention, a network using distributed representations would only have to be able to process inputs representing relations with a maximum arity of five.

6.2 Representing structural systematicity

Structural systematicity refers to the ability of a system to correctly process inputs that it has never encountered before. There are a number of different definitions of structural systematicity. Three levels of systematicity from weak to strong were defined by (Hadley,

1992), but perhaps the most precise definition, using six levels of systematicity has been given by Niklasson and van Gelder (Niklasson & van Gelder, 1994). These levels are:

1. Where no novelty is present in the inputs presented to the system. Every input presented to the system has already been encountered (for example in the training phase of a connectionist system).

2. Where novelty is present in the inputs, but all the individual arguments in these inputs have at some time appeared in the same positions in inputs previously encountered by the system.

3. Where the inputs contain at least one example which has an argument in a position in which it has never before been encountered by the system. For example, a connectionist system could be trained on a selection of input patterns in which a particular variable or constant was never present in the first argument position of the patterns used for training, and then tested on a set of patterns where that variable or constant was present at that argument position.

4. Where novel variables or constants appear in the new inputs presented to the system.

5. Where there is novel complexity in the new inputs presented to the system. To display this form of complexity a connectionist system would have to be capable of being trained on patterns with n inputs in the training set and then correctly process patterns with $n + 1$ inputs in the test set (this is related to the arguments discussed in section 6.1).

6. Where the test set contains novel inputs and novel complexity. Both the fourth and fifth points above would have to appear in the test set.

Symbolic AI variable binding systems can display all these levels of systematicity as they are not dependent on the makeup of a training set to form their representations. Hence they can generate and

process new items whilst only being restricted by physical constraints, such as the size of available memory resources. Localist connectionist systems are not usually dependent on the make-up of a training set to form their representations, and so easily cope with tasks involving up to the third level of systematicity described above, but because of the problems outlined in section 6.1, have problems displaying the fourth and higher levels because any recursive representation in the network has to be constructed to a pre-specified depth, unlike the (potentially almost infinite) recursive representation provided with symbolic systems..

Connectionist systems using distributed representations are heavily dependent on the make-up of the training set used to develop these representations, hence they often do not display systematicity beyond the second level described above. However, in performing simple logical operations with a system using distributed representations (Niklasson & van Gelder, 1994) systematic performance of a neural network both when exposed to novel inputs and when exposed to inputs appearing in novel positions has been demonstrated (i.e. up to the fourth level above).

7. Conclusions

We reviewed here a formalism that facilitates variable binding in connectionist networks. The DN formalism is useful because it encompasses a variety of variations on basic neural network formulations, and thus provides a unified treatment of connectionist mechanisms necessary for variable binding, logical reasoning, and a number of other endeavors. While it is straightforward to implement DN networks directly, such networks can also be easily translated into standard neural networks (Sun, 1992). Thus in a way DN networks serve as an aggregate description of neural network architectures, especially architectures dealing with variable binding. It has been argued that connectionist systems may well offer an opportunity to escape some of the problems of symbolic AI such as brittleness only if ways can be found of instantiating these sources of power displayed by symbolic systems within connectionist systems (Smolensky, 1988; Sun, 1995b; Sun, 1995c). At least in the case of variable binding, connectionist

systems have not as yet matched the power of symbolic AI systems. However any attempt to match the capabilities of a symbolic theorem prover using a connectionist system may be misguided. On the other hand, connectionist systems may give better models of the (limited) variable binding and inference systems for human-like performance and/or greater flexibility. Connectionist systems have some unique advantages, for example they generalise better to unexpected or noisy inputs than symbolic AI systems, and for complex tasks some form of hybrid system (Sun & Bookman, 1994) combining the capabilities of connectionist and symbolic AI components may be appropriate.

New theories of representation are being developed. Some theories attempt to link symbolic AI and connectionist representations (Smolensky *et. al.*, 1992). Others try to subsume both types of representation within a higher level theory (van Gelder & Port, 1995). Hopefully, from these theories will spring new types of more powerful and flexible representation for building better intelligent systems.

The authors

Dr. Antony Browne is a senior lecturer in computer science at London Guildhall University. He received his PhD. In 1995 from South Bank University in computer science. Dr Browne's research interests include the modeling of cognitive processes using neural networks and rule extraction from neural networks for data mining. He is the author of over 20 papers, and has edited or contributed to three books, including editing the two book *Neural Network Perspectives on Cognition & Adaptive Robotics* and *Neural Network Analysis, Architectures & Algorithms* (both published by the Institute of Physics Press, Bristol, UK). He is chair of the International Neural Network Society SIG on Higher Cognitive Modeling.

Dr. Ron Sun is an associate professor of computer science and psychology at the University of Alabama and a visiting scientist at the NEC Research Institute. He received his PhD. in 1991 from Brandeis University in computer science. Dr. Sun's research interest centers around the studies of intelligence and cognition, especially in the areas of commonsense reasoning, human and machine learning, and hybrid

connectionist models. He is the author of over 80 papers, and has written, edited or contributed to 10 books, including authoring the *book Integrating Rules and Connectionism for Robust Commonsense Reasoning* (published by John Wiley and Sons) and co-editing *Computational Architectures Integrating Neural and Symbolic Processes* (published by Kluwer) and *Connectionist-Symbolic Integration* (published by Lawrence Erlbaum Associates). For his paper on integrating rule-based reasoning and connectionist models, he received the 1991 David Marr Award from Cognitive Science Society. He organized and chaired the AAAI Workshop on Integrating Neural and Symbolic Processes, 1992, and the IJCAI Workshop on Connectionist-Symbolic Integration, 1995, as well as co-chairing the AAAI Workshop on Cognitive Modeling, 1996. He was a guest editor of the special issue of the journal *Connection Science* on architectures for integrating neural and symbolic processes and the special issue of the *IEEE Transactions on Neural Networks* and hybrid intelligent models. Dr. Sun serves on the editorial boards *of Connection Science*, *Applied Intelligence*, and *Neural Computing Surveys*.

References

AJJANAGADDE, V. and L. SHASTRI. (1989) Efficient inference with multi-place predicates and variables in a connectionist system. *Proceedings of the 11th Annual Cognitive Science Society Conference*, Hillsdale, NJ. Lawrence Erlbaum, 396-403.

ALEKSANDER, I. (1989) The logic of connectionist systems. In I. Aleksander, editor, *Neural Computing Architectures*. Cambridge, MA. MIT Press,

ANANDAN, P., S. LETOVSKY and E. MJOLNESS. (1989) Connectionist variable binding by optimization. *Proceedings of the 11th Annual Conference of the Cognitive Science Society*, 388-395.

ANDREWS, R., B. TICKLE, M. GOLEA and J. DIEDERICH. (1997) Rule extraction from trained artificial neural networks. In A. Browne, editor, *Neural Network Analysis, Architectures and Algorithms*, Bristol, UK. Institute of Physics Press.

BALLARD, D. H. (1986) Parallel logical inference and energy minimisation. *Proceedings of the AAAI National Conference on Artificial Intelligence*, 203-208.

BARNDEN, J. A. (1989) Neural net implementation of complex symbol processing in a mental model approach to syllogistic reasoning. *Proceedings of the International Joint Conference on Artificial Intelligence*, 568-573.

BARNDEN, J. A. (1994) Complex symbol-processing in conposit, A transiently localist connectionist architecture. In R. Sun and L. Bookman, (Eds.) *Computational Architectures Integrating Neural and Symbolic Processes: A perspective on the state of the art*. Boston, MA. Kluwer.

BROWNE, A. (1996) Measuring distribution in distributed representations. *Neural Processing Letters*, **3**,73-79.

BROWNE, A. (1997) Challenges for neural computing. In A. Browne, editor, *Neural Network Perspectives on Cognition and Adaptive Robotics*. Bristol, UK. Institute of Physics Press.

BROWNE, A. (1998a) Performing a symbolic inference step on distributed representations. *Neurocomputing*, **19**, 23-34.

BROWNE, A. (1998b) Detecting systematic structure in distributed representations. *Neural Networks*, **11**(5) 815-824.

BROWNE, A. and J. PILKINGTON. (1994) Unification using a distributed representation. *Association for Computing Machinery Bulletin on Artificial Intelligence*, **5**(2), 5-7.

BROWNE, A. and J. PILKINGTON. (1995) Performing variable binding with a neural network. In J. Taylor, editor, *Neural Networks*. Henley on Thames, UK. Alfred Waller.

BULLINARIA, J. (1997) Analyzing the internal representations of trained neural networks. In A. Browne, editor, *Neural Network Analysis, Architectures and Algorithms*. Bristol, Institute of Physics Press.

COLLINS, A. and J. LOFTUS. (1975) Spreading activation theory of semantic processing. *Psychological Review*, **82**, 407-428.

DIEDERICH, J. (1988) Connectionist recruitment learning. *Proceedings of the European Conference on Artificial Intelligence*, 351-356.

DIEDERICH, J. (1991) Steps towards knowledge-intensive connectionist learning. In: J. A. Barnden and J. B. Pollack (Eds.), *Advances in connectionist and Neural Computation Theory*, Vol. 1. Norwood, NJ. Ablex.

DYER, M. G. (1990) Distributed symbol formation and processing in connectionist networks. *Journal of Experimental and Theoretical Artificial Intelligence*, **2**,215-239.

DYER, M. G. (1995) Connectionist natural language processing: A status report. In R. Sun and L. A. Bookman, editors, *Computational Architectures Integrating Neural and Symbolic Processes*. Boston, USA. Kluwer.

FELDMAN, J. A. and D. H. BALLARD. (1992) Connectionist models and their properties. *Cognitive Science*, **6**(3).

FRANKLIN, S. and M. GARZON. (1990) Neural computability. In O. Omidvar, editor, *Progress in Neural Networks*, Vol. 1. NJ, USA. Ablex.

HADLEY, R. (1992) Compositionality and systematicity in connectionist language learning. *Proceedings of the 14th Annual Conference of the Cognitive Science Society*, 659-664.

HALFORD, G. S. (1993) Creativity and the capacity for representation: Why are humans so creative? *AISB Quarterly*, **85**, 32-41.

HINTON, G. E. (1990) Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, **46**, 47-75.

HINTON, G. E., J. L. McCLELLAND and D. E. RUMELHART. (1986) Distributed representations. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, Vol. 1. Cambridge, MA. MIT Press.

HOLLDOBLER, S. (1990) CHCL - a connectionist inference system for Horn logic based on the connection method. *Technical Report TR-90-042*, International Computer Science Institute, Berkeley, CA.

KOHONEN, T. (1988) *Self-Organization and Associative Memory*. New York. Springer Verlag.

KURFESS, F. (1991) Unification on a connectionist simulator. *Proceedings of the International Conference on Artificial Neural Networks*, 471-476.

LANGE, T. and M. G. DYER. (1989) High-level inferencing in a connectionist network. *Technical Report UCLA-AI-89-12*, UCLA, Los Angeles.

LEE, G., M. FLOWERS and M. G. DYER. (1990) Learning distributed representations for conceptual knowledge and their application to script-based story processing. *Connection Science*, **2**(4), 313-345.

LOOKE, P. R. V. (1995) QNET: A quantum mechanical neural network. A new connectionist architecture and its relevance for variable binding and constraint satisfaction problems. *Cybernetica*, **38**(1), 85-106.

MARKOV, Z. (1991) A tool for building connectionist-like networks based on term unification. *Proceedings of the Processing Declarative Knowledge International Workshop*, 199-213.

MINSKY, M. (1983) A framework for representing knowledge. In J. Hughland, editor, *Mind Design*. Cambridge, MA. MIT Press.

MJOLNESS, E., E. GINDI, and P. ANANDAN. (1989) Optimization in model matching. Neural Computation, **1**, 218-229.

NEWELL, A. (1980) Physical symbol systems. *Cognitive Science*, **4**, 135-183.

NIKLASSON, L. and T. van GELDER. (1994) Can connectionist models exhibit non-classical structure sensitivity? *Proceedings of the Cognitive Science Society*, 664-669.

PARK, N. S., D. ROBERTSON and K. STENNING. (1995) Extension of the temporal synchrony approach to dynamic variable binding in a connectionist inference system. *Knowledge-Based Systems*, **8**(6), 345-357.

PINKER, S. and A. PRINCE. (1988) Language and connectionism. In S. Pinker and J. Mehler, editors, *Connections and Symbols*. Cambridge, MA. MIT Press.

PLATE, T. A. (1991) Holographic reduced representations. *Technical Report CRG-TR-91-1*, Department of Computer Science, University of Toronto, Ontario, CA.

PLATE, T. A. (1994) Distributed representation and nested compositional structure. PhD thesis, Department of Computer Science, Toronto, CA.

POLLACK, J. B. (1990) Recursive distributed representations. *Artificial Intelligence*, **46**,77-105.

ROSENFELD, R. and D. TOURETZKY. (1988) Coarse coded symbol memories and their properties. *Complex Systems*, **2**, 463-484.

RUMELHART, D. E., G. E. HINTON and R. J. WILLIAMS. (1986) Learning internal representations by back-propagating errors. *Nature*, **323**, 533-536.

SHARKEY, N. (1992) The ghost in the hybrid - a study of uniquely connectionist representations. *AISB Quarterly*, **79**, 10-16.

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

SMOLENSKY, P. (1988) On the proper treatment of connectionism. *Behavioral and Brain Sciences*, **11**, 1-74.

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

SMOLENSKY, P., G. LEGENDRE AND Y. MIYATA. (1992) Principles for an integrated connectionist and symbolic theory of higher cognition. *Technical Report CU-CS-600-92, Computer Science Department*, University of Colorado, Boulder.

STOLKE, A. (1989) A connectionist model of unification. *Technical Report TR-89-032*, International Computer Science Institute, Berkley, CA.

SUN, R. (1989) A discrete neural network model for conceptual representation and reasoning. *Proceedings of the 11th Conference of the Cognitive Science Society*, 916-923.

SUN, R. (1992) On variable binding in connectionist networks. *Connection Science*, **4**, 93-124.

SUN, R. and L. A. BOOKMAN. (1994) *Computational Architectures Integrating Neural and Symbolic Processes: A Perspective on the State of the Art*. Boston, MA. Kluwer.

SUN, R. (1995a) Schemas, logic and neural assemblies. *Applied Intelligence*, **5**, 83-102.

SUN, R. (1995b*) Integrating Rules and Connectionism for Robust Commonsense Reasoning*. New York, John Wiley & Sons.

SUN, R. (1995c) Robust reasoning: integrating rule-based and similarity-based reasoning. *Artificial Intelligence*, **75**(2), 241-296.

TOMABECHI, H. AND H. KITANO. (1989) Beyond PDP: The frequency modulation neural network architecture. *Proceedings of the International Joint Conference on Artificial Intelligence*, 186-192.

TOURETZKY, D. S. and G. E. HINTON. (1988) A distributed connectionist production system. *Cognitive Science*, **12**(3), 423-466.

van GELDER, T. (1991) What is the ‘D’ in ‘PDP’? A survey of the concept of distribution. In W. Ramsey, S. Stich, and D. E. Rumelhart, editors, *Philosophy and Connectionist Theory*. Hillsdale, NJ. Lawrence Erlbaum.

van GELDER, T. and R. F. PORT. (1995) It’s about time: An overview of the dynamical approach to cognition. In R. F. Port and T. van Gelder, editors, *Mind as motion: Explorations in the dynamics of cognition*. Cambridge, MA. MIT Press.

WEBER, V. (1992) Connectionist unification with a distributed representation. *Proceedings of the International Joint Conference on Neural Networks*, 555-560.

WEBER, V. (1993) Unification in Prolog by connectionist models. *Proceedings of the Fourth Australian Conference on Neural Networks*, A5-A8.

Figure Headings.

Fig 1. Implementing first-order rules in a connectionist network.

Fig 2. A connectionist network for backward-chaining reasoning where “+” indicates enabling and “-” indicates blocking.

Fig 3. A DN node.

Fig 4. The interconnections in an assembly.

Fig 5. An example network.

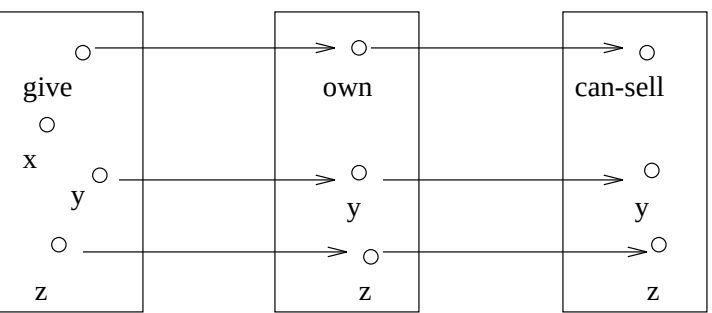
Fig 6. Phase synchronization for variable binding.

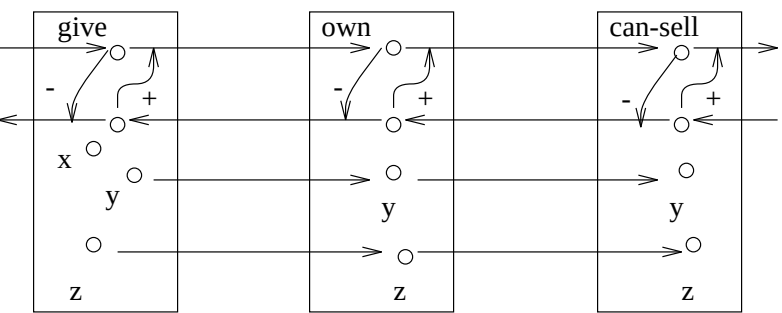
Fig 7. A network utilizing phase synchronization for variable binding.

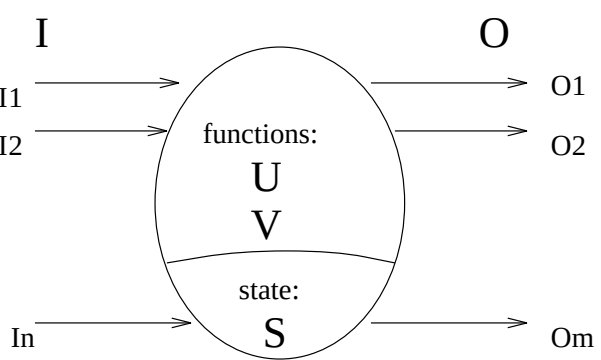
Fig 8. Example symbolic level representation of term pairs presented to input autoassociator.

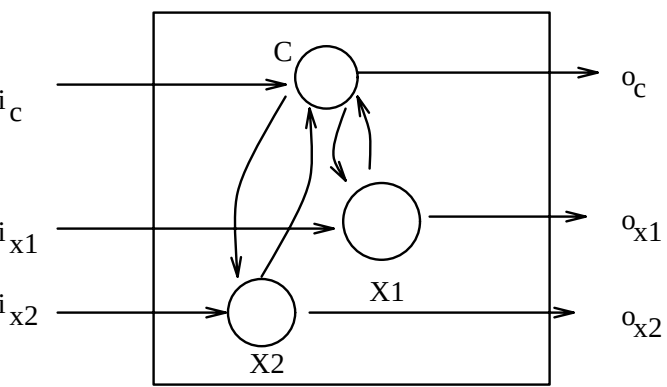
Fig 9. Example symbolic level representation of unification results presented to output autoassociator.

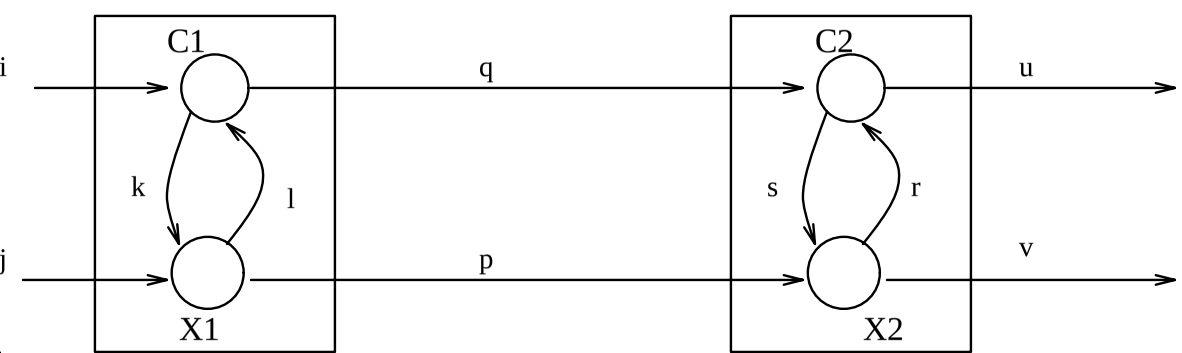
Fig 10. Four layer network maps distributed representation of pre-unification term pairs from input autoassociator to distributed representation of unification results on target autoassociator.

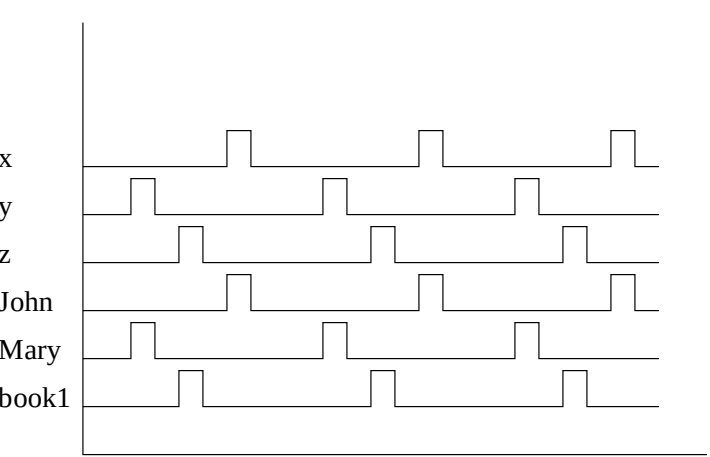

give
own
can-sell
x
y
y
y
z
z
z

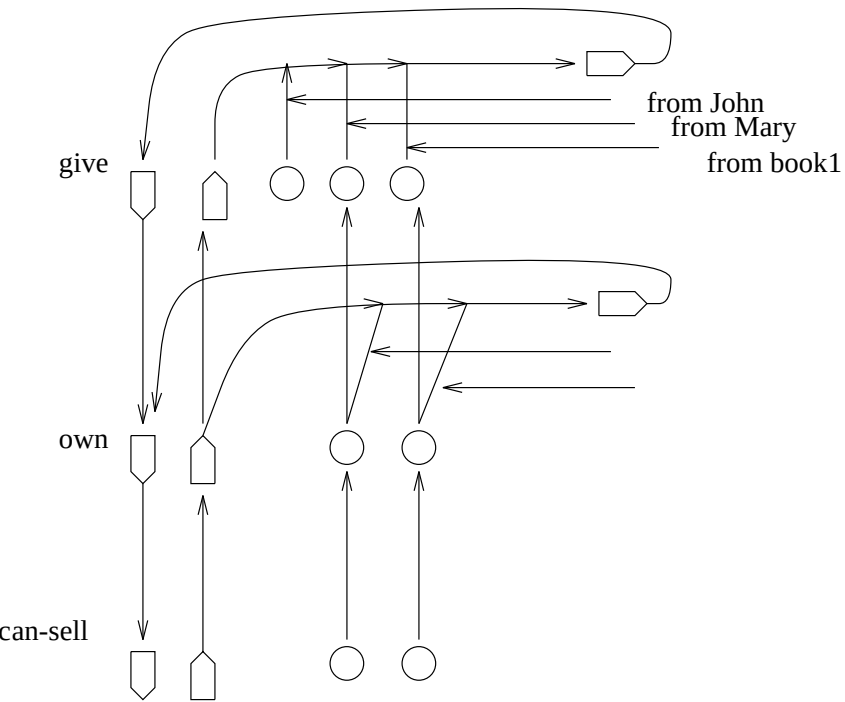

give
own
can-sell
-
+
-
+
-
+
x
y
y
y
z
z
z

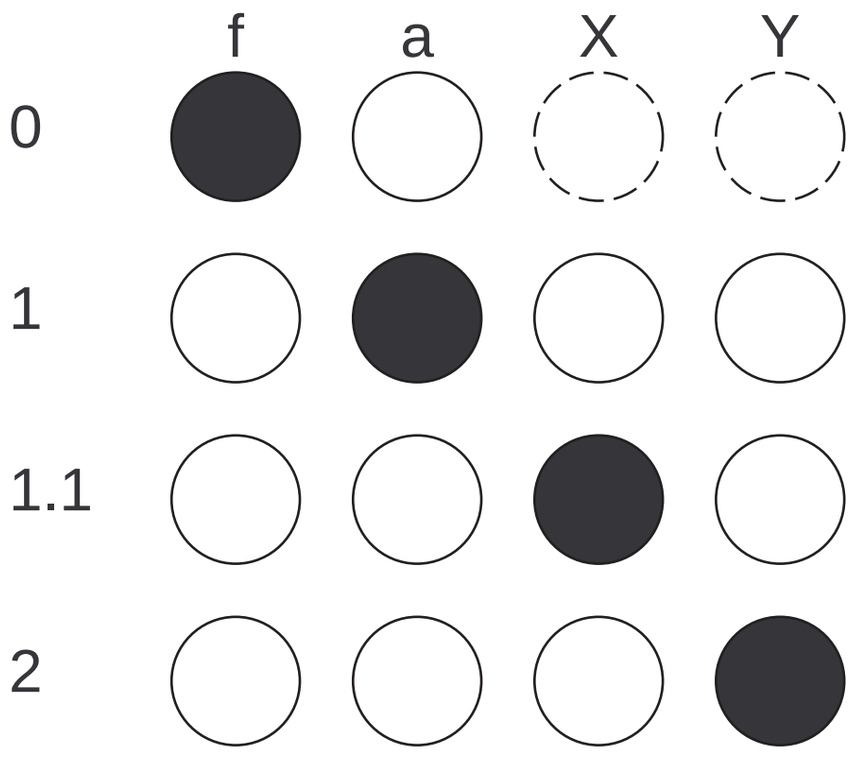

I
O
I1
I2
In
functions:
U
V
state:
S
O1
O2
Om


C
i_c
o_c
i_{x1}
o_{x1}
X1
i_{x2}
o_{x2}
X2

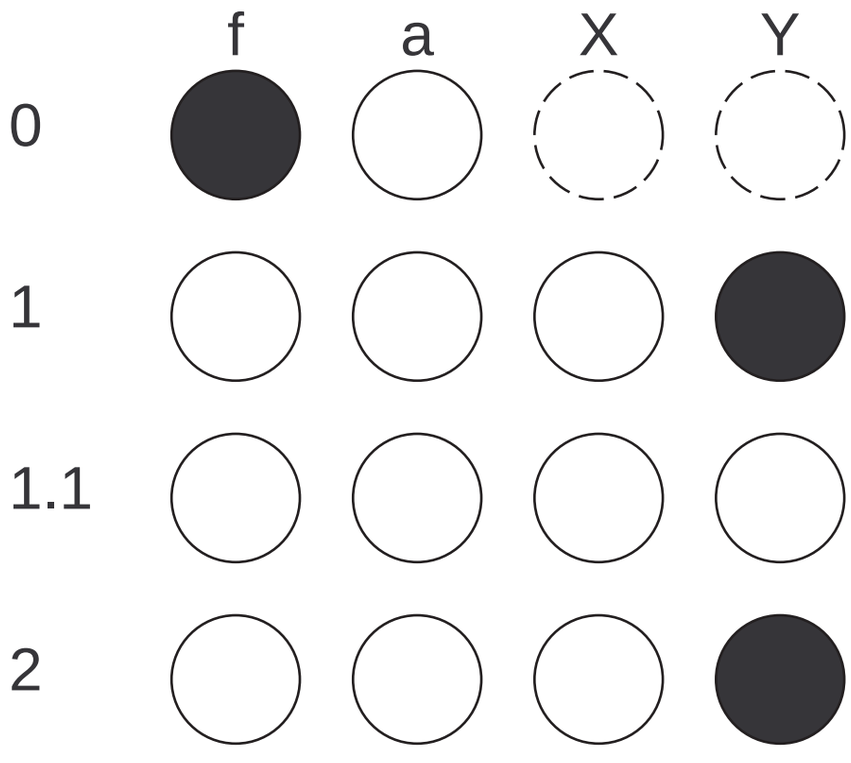

C1
C2
i
q
u
k
l
s
r
j
p
v
X1
X2


x
y
z
John
Mary
book1

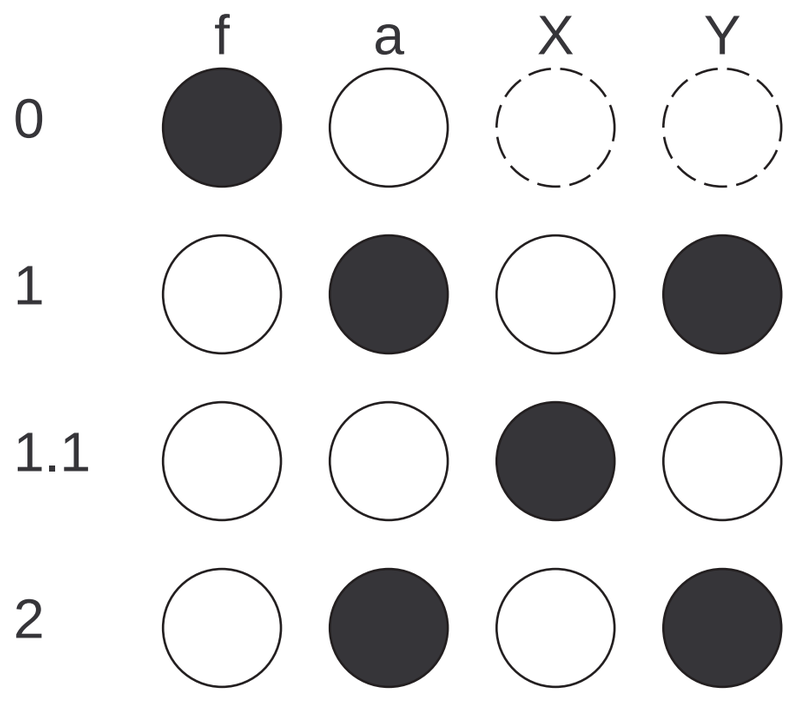

from John
from Mary
from book1
give
own
can-sell



First pre-unification term f(a(X),Y)

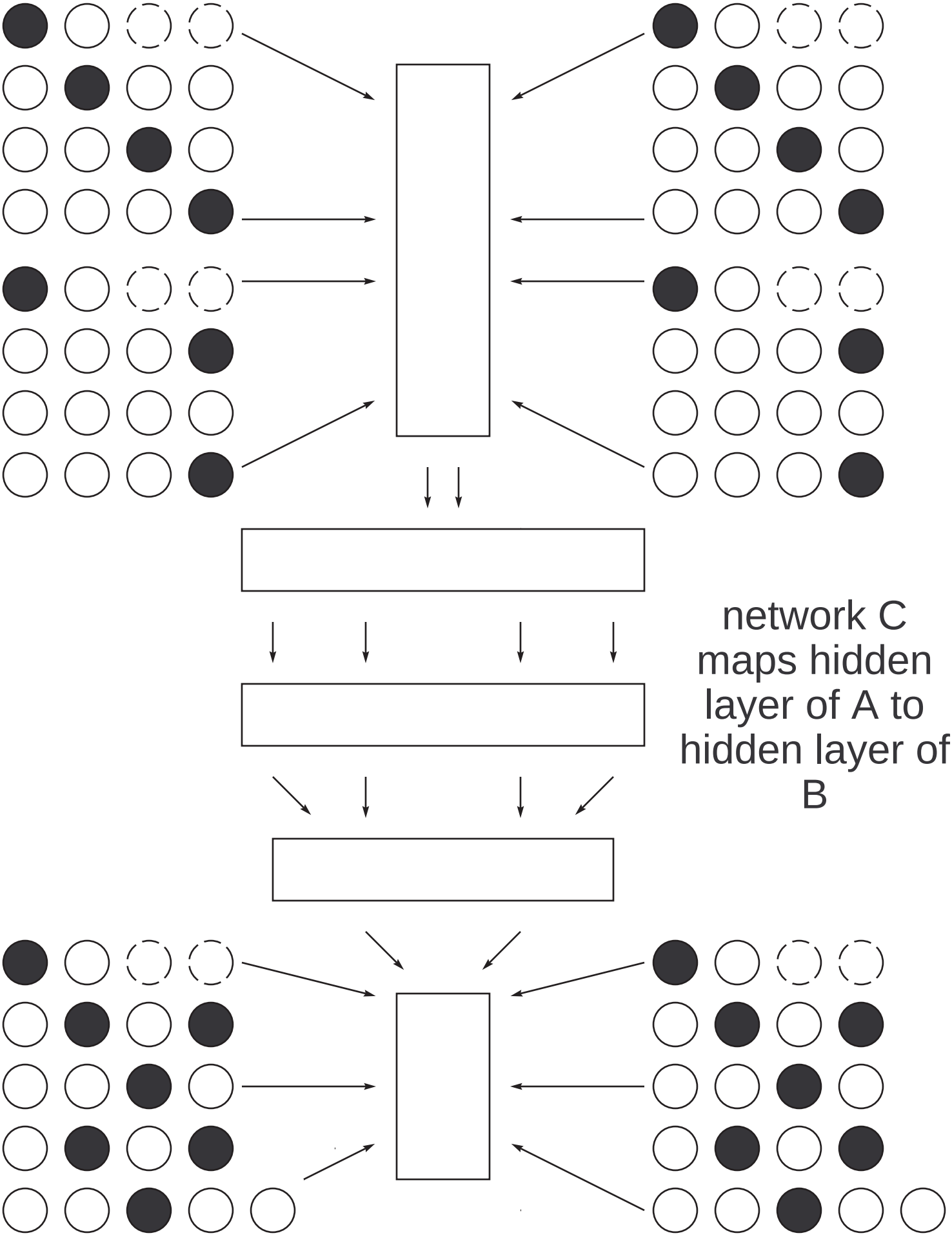


Second pre-unification term f(Y,Y)



occurs check unit

Unification results f(a(X),a(X))


network C maps hidden layer of A to hidden layer of B
network B autoassociates unification results