UC Merced

Proceedings of the Annual Meeting of the Cognitive Science Society

Title

The Processing of Associations versus the Processing of Relations and Symbols : A Systematic Comparison

Permalink

https://escholarship.org/uc/item/3jx321d4

Journal

Proceedings of the Annual Meeting of the Cognitive Science Society, 17(0)

Authors

Phillips, Steven

Halford, Graem S.

Wilson, William H .

Publication Date

1995

Peer reviewed

The Processing of Associations versus the Processing of Relations and Symbols: A Systematic Comparison

Steven Phillips
Department of Computer Science
University of Queensland
Brisbane 4072 Australia
stevep@etl.go.jp

Graeme S. Halford
Department of Psychology
University of Queensland
Brisbane 4072 Australia
gsh@psych.psy.uq.oz.au

William H. Wilson
School of Computer Science & Engineering
University of New South Wales
Sydney 2052 Australia
billw@cse.unsw.edu.au

Abstract

A mathematical basis is proposed for the distinction between associative and relational (symbolic) processing. Associations can be contrasted with relations in terms of ordered pairs versus general ordered N-tuples, and unidirectional access versus omnidirectional access. Relations also have additional properties: they can exhibit predicate-argument bindings, they can be arguments to higher-order structures, and they can participate in operations of selection, projection, join, union, intersection, and difference. Relations can be used to represent structures such as lists, trees and graphs, and relational instances can be thought of as propositions. Within neural net architectures, feedforward networks can be identified with associative processing, and tensor product networks with relational processing. Relations have the essential properties of symbolic processing; flexibility, accessibility, and utility for representing complex data structures.

Introduction

We propose a mathematical basis for the distinction between associative and relational processing. We suggest this distinction captures much of the meaning contained in the distinction between traditional associationism and symbolic processing, and has implications for neural network modelling of psychological tasks.

We illustrate our argument using the balance-scale (which balances when the product of weights and distances on the two sides are equal, that is when $W_l D_l = W_r D_r$). A common form of assessment has been to ask participants to predict the balance state (whether the beam will balance, or which side will go down) when various combinations of weights are placed at various distances from the fulcrum (Siegler, 1981). Other assessments require participants to specify the weight or distance on one side that will balance a particular combination of weight and distance on the other side (Surber & Gzesh, 1984) To demonstrate understanding, a participant must be able to retrieve any variable, given the rest (e.g. given W_l, D_l, W_r, and the outcome BALANCE it should be possible to specify D_r; given W_l, D_l, W_r, and D_r, it should be possible to specify the state of balance). Thus we say that access to a relational concept should be *omnidirectional*.

An example of an association would be a rabbit running on seeing a *fox*, that is; *fox* → *run*. It is a link between two elements, *fox* and *run*. It is unidirectional, because running does not automatically activate a representation of *fox*.

Associative Versus Relational Processing

In this section, we give an abstract description of the associative and relational modes of processing.

Associative Processing

Data Structures

In an associative system we assume the existence of a set of symbols $S = \{a_1, a_2, ..., a_N\}$. The primary data structure of an associative system is a set of pairs of symbols A, denoted: $A = \{(a_i, a_j) \mid a_i, a_j \in S \text{ and } a_i \text{ cues } a_j\}$

Operations

There are three basic operations in an associative system:
- *Cue* - takes the symbol a_i and returns its associated symbol a_j, from the set of pairs in A. $F_{cue}(a_i) \rightarrow a_j$
- *Form association* - takes two symbols a_i and a_j and adds the pair to the set A to form a new set A'.
  $F_{assoc}(a_i, a_j) \rightarrow A' = A \cup \{(a_i, a_j)\}$
- *Delete association* - takes a pair and removes it from the set A to form a new set A'.
  $F_{del\_a}(a_i, a_j) \rightarrow A' = A \setminus \{(a_i, a_j)\}$

The basic symbols may themselves be composed of other symbols (i.e., they are not necessarily atomic). In this way, associations may be formed between three or more basic symbols, or between complex representations.

Relational Processing

Data structures

A relational system consists of a set of relations $\{R_i\}$ where each relation R_i (of *arity*, i.e. number of components, $n(i)$), corresponds to a set of $n(i)$-tuples:
$R_i = \{(x_1, ..., x_{n(i)}) \in S_{i1} \times ... \times S_{in(i)} \mid R_i(x_1, ..., x_{n(i)}) \text{ holds}\}$

For example, if $S_1 = \{john, mary, tom\}$, and S_2 is the set of natural numbers, then a relation has-age on $S_1 \times S_2$ might be written as the set of pairs $\{(john, 24), (mary, 22), (tom, 3)\}$.

A relation can also be conceptualised as a table. For example, the relations "has-age" and "loves" may be represented by the following tables:

Predicate	Object	Years
has-age	john	24
has-age	mary	22
has-age	tom	3

Predicate	Person	Object
loves	john	mary
loves	john	sue
loves	mary	tom

It is redundant to store the predicate name as a separate column in the table, however, we have adopted this convention to allow for situations in which the arguments act as cues for accessing the predicate.

Each row of the table is a tuple from the set which formally constitutes the relation. It is also convenient to write a relation in terms of its attributes - that is, the types of the arguments it takes. Thus, R<s> denotes a relation R with attribute sequence $s = A_1, ..., A_n$. For example, the binary relation 'has-age' can be identified as: has-age<object,years>.

Relations are a general purpose data structure, and in fact can be used to represent other commonly used data structures, such as lists, trees and graphs. For example, the list of objects $L_1 = $ [john, mary, tom] can be represented by the relation:

is-list-of<list, head, tail> = {(L1, john, L2), (L2, mary, L3), (L3, tom, nil)}. (Here *nil* is a constant representing an empty list.) Furthermore, relational instances (i.e., rows of a table) have an assigned truth value (i.e., TRUE), and so can be thought of as propositions. For example, larger-than(whale,man) is a proposition that is TRUE.

Operations

We provide informal definitions of relational operators:

- F_{select} - given a relation and one or more components, returns the row(s) with those components.
  Example 1: F_{select}(has-age,Person=tom) → (tom,3).
- $F_{project}$ - given a relation and one or more attributes (column names), returns those columns of the table.
  Example 2: $F_{project}$(loves, Person) → {john, john, mary}.
- F_{join} - takes two tables and returns a new table joined at common components in specified columns.
  Example 3: F_{join} could be used to "paste together" the relation has-age, defined above, with a relation has-height = {(john, 175), (mary, 165), (tom, 95)} (where the second component is height in centimetres), joining at the Person column, to produce a new relation age-and-height = {(john, 24, 175), (mary, 22, 165), (tom, 3, 95)}.
  Example 4: Consider the relation smaller-than<animal1, animal2> = {(mouse, cat), (cat, dog), (dog, horse)}. A more complicated form of F_{join} could be used to "paste together" one instance of relation smaller-than, joining at the animal2 column, with another instance of the same relation smaller-than, joining at the animal1 column, to produce a new relation (call it "much-smaller-than") =

{(mouse, dog), (cat, horse)}. This example composes the relation with itself, much as in transitive inference.

In addition, there are set-like operators, F_{union}, $F_{intersection}$, and $F_{difference}$, which form the set-theoretic union, intersection and difference of two relations with the same attribute lists, and operators F_{add_r}, F_{del_r}, and F_{update_r}, for adding, deleting, and updating relational instances (Codd, 1990).

Comparison Between Associative and Relational Modes

Both associative and relational systems utilise links between component symbols to construct more complex symbols. However, beyond this similarity there are significant differences in their processability.

Compositionality: Associations are limited to pairs of symbols, whereas relations can be between arbitrarily many symbols (including pairs). Furthermore, associations do not recognise a predicate (a symbol for the relationship between other symbols). Relations on the other hand are predicated. This is a crucial distinction between associations as implicit representations (representations that are not available to other processes), and relations as explicit representations (relations that are available to other processes). The predicate is the explicitation of links between symbols. Because a relation is explicitated via the predicate symbol, relations are available as arguments to other relations. Associations may cue other associations through chaining, but they are not themselves available for association. Relations, on the other hand, can exist between other relations. For example, the relation "because" can take the unary relation "cried" and the binary relations "kissed" as arguments: because(cried(Tom), kissed(John,Mary)).

Directionality: Associative systems are unidirectional: the first component can cue the second, but not the reverse. Relations, however, are in general omnidirectional. Any subset of components can be used to access the remaining components. The number of returned relational instances will, of course, depend on the uniqueness of the supplied components.

Structure sensitivity: Associative systems are structurally very weak. An associative system can be conceptualised as a single table of two columns. Relational systems, by contrast, are stronger structurally. Components are accessible purely by their roles. Furthermore, new relations can be created on the basis of structural operations. A relation R can be used to generate its inverse by using the project operator (i.e., $R^{-1} = F_{project}$(R<A1,A2>, A2, A1)). For example, > (greater-than) can be mapped to < (less-than), by swapping arguments, without having to learn an entirely new table of associations. In other words, relational systems have the capacity to create *virtual* tables, thus circumventing extensive retraining.

Implications

The data structures and operations defined in the previous section have a number of implications in terms of resources (both time and space) for models utilising these two modes of processing. Suppose we a relation R<subject, relation,

689

object> includes the concepts "John loves Mary" and "Sue loves Tom", from which there is sufficient information to correctly answer questions such as: "Who loves Mary?"; Who does Sue love?"; and "What is the relationship between Sue and Tom?".

An associative system, where the only complex data structure is a pair, must construct new symbols in order to correctly answer these questions. For example, (loves-Mary, John), (Sue-beloved, Tom), and (Sue-Tom-relation, loves). Each question is matched against the first component of each pair. The pair with the closest match triggers the second component resulting in the correct answer. In fact, each concept requires three associations, one for each possible response (e.g., loves, John, Mary).

A relational system, using the same two concepts, only requires a single table with two entries: (loves,John, Mary), and (loves,Sue, Tom). The select and project operators are sufficient to extract any combination of components. For example,

$$F_{project}(\quad F_{select}(R, <relation=loves,object=Mary>),$$
$$subject) = John;$$
$$F_{project}(\quad F_{select}(R, <(subject,John),(object,Mary)>),$$
$$relation) = loves.$$

In the relational case, only one entry per concept is required. Thus, a relational system uses less space than an equivalent associative system. However, the associative system only makes one match to the left-hand side of each pair, whereas the relational system must align the input cues with the appropriate relational components. The reduction in space is at the expense of additional processing required to perform some sort of structural alignment in the relational system.

The flexibility of relational operators relates to a further implication which is generalisation. Suppose we have the concept: "a whale is larger than a horse", which implies the related concept "a horse is smaller than a whale". In an associative system both concepts require separate learning steps (i.e., use of the F_{assoc} operator). The relational system, by contrast, need only be trained on the first instance. Since the inverse relation can be constructed dynamically via the project operator, it is not necessary to have also been trained on the second concept for subsequent processing. Thus, relations have a greater degree of generalisation.

Neural Network Implementations

This section suggests neural network architectures that can readily be used to implement systems with similar properties to the associative and relational systems that we have described in the preceding section. Of course, neural network architectures may be of enormous variety, and we do not mean to suggest that these are the only NN architectures that match associative/relational systems (see for instance Hinton (1988)).

Feedforward networks (FFN)

FFN (for example, Rumelhart, Hinton, & Williams, 1986) have two modes of operation: (1) processing mode - input activations are propagated throughout the network resulting

in output activations; and (2) learning mode - error signals are propagated throughout the network resulting in changes to connection weights. The first mode corresponds to the F_{assoc} operator. The network returns a vector which is some (possibly non-linear) combination of matched first argument vectors. The second mode corresponds to the F_{assoc} and F_{del_a} operators by adding/deleting new input-output pairs encoded as weighted connections.

FFNs have demonstrated generalisation in the sense that component symbols can be brought together to form a new symbol at the hidden layer of units. It is not necessary for the network to be trained on all combinations (see, for example, Phillips & Wiles, 1993). However, this form of generalisation is different from generalisation to all functions implied by a relation. A FFN, in general, cannot demonstrate this degree of generalisation and therefore is not a relational processor.

To illustrate this point, consider the simple case of representing and processing the following binary relation: $R_2 = \{(a,b)\}$. Derivable from this relation are two functions: $f_1(a) = b$, and $f_2(b) = a$. To completely represent R_2, an FFN must implement the second order function:

$$F(f_1, a) = b; \quad F(f_2, b) = a.$$

For this simple relation R_2, the co-domain of F can be completely determined by the second arguments (i.e., a and b) only. Dropping the first arguments for simplicity, the FFN must implement the function:

$$F(a) \rightarrow b; \quad F(b) \rightarrow a.$$

Since, in general, there is no similarity between a and b, the FFN must be trained on both instances of the function F before the network is guaranteed to implement F, in the sense of approximating F at above-chance level over repeated trials, and therefore, representing R_2. Hence, a standard FFN implementation of relational representations necessitates being trained on all implicated functions.

The second crucial point is that the relationship between component symbols is encoded as weighted connections and activation functions between groups of units. These weights (encodings) are, in general, not available to other processes within the FFN, although they may be analyzed via external processes such as cluster analysis or principal components analysis. In other words, the predicate (i.e., the name for the set of pairs) is implicit to the system, not explicit as in a relational system. The properties of unidirectionality and implicit predication identify the FFN as an associative system.

So FFNs can be used to implement an association in a natural way.

Tensor Networks (TN)

In a tensor network, predicate and arguments are bound together via an organisation of unit connectivity that implements the outer product operator (Halford, Wilson, Guo, Gayler, Wiles, & Stewart, 1994), which contrasts with Smolensky's (1990) use of the tensor where role and filler components are bound together via the outer product. The arguments are supplied as inputs, and weights are updated as the outer product of the two inputs. This allows for either a or b, applied to the right set of input units to

map to the other vector (by implementing the inner product operator).

Mathematically, the learning process can be thought of as a function TL (for tensor learning), which, given a tensor network F and (say) a pair to learn, (a,b), returns a modified version F' of F that "knows" the pair (a,b): TL(F, (a,b)) → F, where F(a,_) → b, F'(_,b) → a.

The important point is that T need only be applied once to represent both implicated functions. Thus, at one crucial point the tensor network operates in the relational mode of processing. Secondly, since the predicate-argument bindings can be explicitly represented as an activation tensor, this tensor can also be used as an argument to other tensors (via the outer product), thereby implementing relations between relations. Thus, the properties of omnidirectionality and generalization to implicated functions identifies the tensor network as an implementation of a relational system.

Psychological tasks

McClelland (1995) has modelled human performance on the balance scale as a three-layered network. There are input units which code the weights and distances on the left and right, a hidden layer, and output units which code the balance-state. The model gives a good fit to human performance in predicting the balance state, and captures the important torque difference effect (the size of the difference in torque between left and right affects judgment). However the model computes only one function: given W_l, D_l, W_r and D_r it computes the balance-state.

The balance scale can also be represented as a tensor product of five vectors representing the balance state, and each of the input variables, W_l, D_l, W_r, D_r (Halford, 1993). With this representation, any variable can be output, given the remaining variables as input (given any four of W_l, D_l, W_r, D_r and balance-state, the remaining variable can be determined). This model is relational, and gives omni-directional access, whereas the three-layered net is associative, and gives uni-directional access.

The three-layered network does not represent human understanding of the balance scale, because we would be reluctant to attribute complete understanding to a person who could compute only one function in this situation. There are of course other considerations which favor the feedforward network model (e.g. simulating the torque difference effect), and we would not contend that either model is necessarily superior at this stage. However the example neatly illustrates an important difference between associative and relational models of cognitive tasks.

Conclusion

Relations capture many of the properties of symbolic thought. Whereas associations are unidirectional, access to relations is omnidirectional, and so relations have the flexibility characteristic of symbolic thought. There are predicates explicitly representing relations, making them accessible to other cognitive processes, whereas there is no explicit symbol for an associative link. Relations can represent symbolic structures such as propositions, lists,

trees and graphs. We have argued that feedforward networks can be used in a natural way to implement associative modes of processing, and that relational modes of processing can be implemented using tensor product networks. We have characterized the difference between associations and relations and their processing in terms of mathematical properties, and linked each to suitable neural network architectures, and psychological tasks.

References

Codd, E.F. (1990). *The Relational Model for Database Management: Version 2*. Addison-Wesley.

Halford, G.S. (1993). *Children's understanding: the development of mental models*. Hillsdale, N. J.: Erlbaum.

Halford, G.S., Wilson, W.H., Guo, J., Gayler, R.W., Wiles, J., & Stewart, J.E.M. (1994). Connectionist implications for processing capacity limitations in analogies. In K. J. Holyoak & J. Barnden (Eds.), *Advances in connnectionist and neural computation theory, Vol. 2: Analogical connections* (pp. 363-415). Norwood, NJ: Ablex.

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

McClelland, J.L. (1995). A connectionist perspective on knowledge and development. In T. Simon & G. S. Halford (Eds.), *Developing Cognitive Competence: New Approaches to Cognitive Modelling*. Hillsdale, NJ: Erlbaum.

Phillips, S., & Wiles, J. (1993). Exponential Generalizations from a Polynomial Number of Examples in a Combinatorial Domain. In *Proceeedings of the International Joint Conference on Neural Networks*, (pp. 505-508). Nagoya, Japan:

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In D. E. Rumelhart & J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (pp. 318-362). Cambridge: MIT Press.

Siegler, R. S. (1981). Developmental sequences within and between concepts. *Monographs of the Society for Research in Child Development*, 46, 1-84.

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

Surber, C. F., & Gzesh, S. M. (1984). Reversible operations in the balance scale task. *Journal of Experimental Child Psychology*, 38, 254-274.