Symbols Among the Neurons: Details of a Connectionist Inference Architecture

David S. Touretzky
Geoffrey E. Hinton

Computer Science Department
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

Pattern matching and variable binding are easily implemented in conventional computer architectures, but not necessarily in all architectures. In a distributed neural network architecture each symbol is represented by activity in many units and each unit contributes to the representation of many symbols. Manipulating symbols using this type of distributed representation is not as easy as with a local representation where each unit denotes one symbol, but there is evidence that the distributed approach is the one chosen by nature. We describe a working implementation of a production system interpreter in a neural network using distributed representations for both symbols and rules. The research provides a detailed account of two important symbolic reasoning operations, pattern matching and variable binding, as emergent properties of collections of neuron-like elements. The success of our production system implementation goes some way towards answering a common criticism of connectionist theories: that they aren't powerful enough to do symbolic reasoning.

1. Introduction

Computer scientists and others have long been interested in neural network architectures as a means of exploring the question of intelligence. In the past these architectures have been successfully applied to relaxation problems such as those found in low level vision (Marr and Poggio, 1979; Barrow and Tenenbaum, 1981; Ballard *et al.*, 1983); they have also served as the basis for various pattern recognition and associative memory schemes proposed throughout the years (Minsky and Papert, 1969; Post, 1969; Hinton and Anderson, 1981). Recently, a movement within AI and cognitive science called "connectionism" has arisen to investigate massively parallel representations built from simple homogeneous elements as models for higher level cognitive processes. Examples include finding the correct reference frame for object recognition (Hinton, 1981), a psychologically plausible theory of word recognition (McClelland and Rumelhart, 1981), and a mechanism for context-based word sense disambiguation (Cottrell, 1984).

To implement the highest level of cognitive functioning, the one responsible for general reasoning, requires some sort of symbolic inference architecture. On a conventional computer this might be provided by a Lisp interpreter, a resolution theorem prover, or a production system interpreter, all three of which have certain operations in common, namely pattern matching and variable binding. On a connectionist architecture these operations can be difficult to implement, especially if distributed representations are used. Ballard and Hayes (1984) have suggested one way of performing unification in a connectionist network, but they use a local representation.

In a distributed representation each symbol is represented by activity in many units and each unit contributes to the representation of many symbols. Manipulating symbols this way is not as easy as with a local representation where each unit denotes one symbol, but there is evidence that the distributed approach is the one chosen by nature. A key problem, then, is how pattern matching and variable binding can be achieved in systems that use distributed representations. In answer to this problem we present two simple production system interpreters implemented as neural networks, in which distributed representations are used for both the working memory elements and the production rules. The research provides a detailed account of pattern matching and variable binding operations as emergent properties of collections of neuron-like elements.*

2. Two Production Systems

The type of production system we consider here consists of a working memory that contains triples of symbols and a set of production rules that reference this memory. Each rule has a left hand side that matches a pair of working memory triples and a right hand side that specifies any number of triples to be added to or deleted from working memory if the rule should fire. Variables may appear on the left hand sides of rules, where they act as constraints on the match process, and on the right hand sides where their values are instantiated to define the actions the rules take during firing. Our first production system interpreter did not permit variables in the rules; it was conceptually very close to a finite state machine. A sample production rule from this interpreter is shown below. The rule states that if the triples (F A A) and (F B B) are present in working memory, then we may replace them with the triple (G A B).

```
Rule-1:  (F A A)  (F B B)  -->
           +(G A B)  -(F A A)  -(F B B)
```

*The binary threshold computing elements featured in connectionist models are commonly referred to simply as "neurons," but the use of this term by us and by most other connectionist researchers should be understood as metaphorical. Connectionist models are not intended to be physiologically correct in all their detail (they rarely are); rather, they should be computationally interesting and/or psychologically plausible.

We implemented this interpreter as a neural network simulated on a Symbolics 3600 Lisp Machine with about 7000 binary threshold units; the weights and thresholds were constrained to be small signed integers. The particular values used for weights and thresholds of the various cell types are all predefined program constants; knowledge about rules is stored in the connection patterns of the neurons rather than in the weights. The contents of working memory are encoded in the states (either on or off) of working memory cells.

Our second production system interpreter is similarly specialized. It accepts rules where a variable appears in the first position of both triples on the left hand side of each rule, and optionally in the first position of right hand side triples. This system interprets rules such as the following:

```
Rule-2:  (=x A B)  (=x C D)  -->
            +(=x E F)  +(P D Q)  -(=x S T)
```

Here, the appearance of =x in both left hand side triples means the rule can match pairs such as (F A B) and (F C D), but not pairs such as (F A B) and (G C D). Our second interpreter, written as an extension of the first, uses about 8000 units and a completely different set of weights.

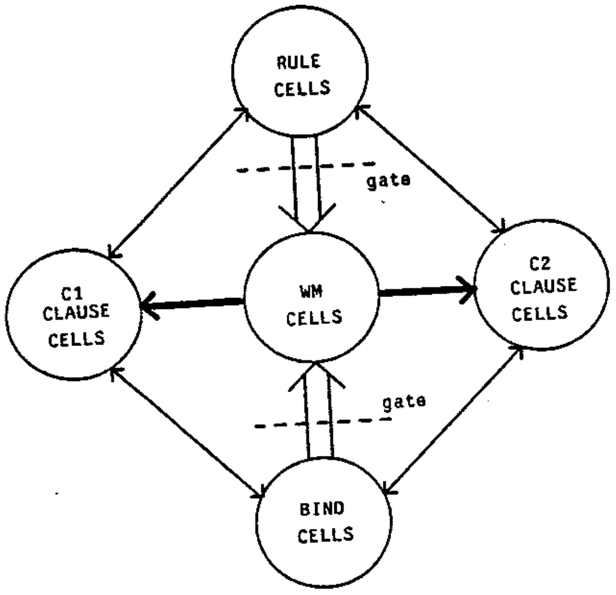

Figure 1: Schematic diagram of our second production system interpreter.

3. Architecture of the Interpreters

Figure 1 is a schematic diagram of our second production system interpreter, which is composed of five "spaces" of cells. (The first interpreter resembles the second except it is missing the Bind cell space.) The central space, labelled WM, is the working memory; it provides inputs to two clause spaces labelled C1 and C2. The clause spaces both influence and are influenced by two other spaces; one of these represents the production rules, while the other implements variable binding and is independent of specific rules. The system in figure 1 is known as a "two stroke production system engine" because it alternately performs each half of the classic production system recognize-act cycle. During the recognize stroke, WM cells exert influence on C1 and C2 cells and a relaxation algorithm is applied to cells in the C1, C2, Rule and Bind spaces until they settle into a state indicating a match. Then, during the act stroke, a set of gated connections from the Rule and Bind cells to the WM cells is opened, allowing the rule that fires to update the contents of working memory.

3.1. Working Memory

Working memory elements are triples of symbols. We have chosen an alphabet size of 25 symbols, giving 25^3 or 15,625 possible triples. Of these, only about half a dozen will be present in working memory at any one time. The most straightforward representation for working memory would be a "local" one, where each possible triple is represented by a specific neuron. Then a neuron in the active state would indicate that the corresponding triple was present. We have rejected this idea in favor of a distributed representation known as coarse coding (Hinton, 1981; Hinton *et al.*, 1985) for two reasons. First, local representations require too many neurons and too many connections; they quickly succumb to combinatorial explosion as the alphabet size or the length of a sequence increases. Neurons are not used efficiently this way; in the system we are describing, with six items in working memory, only about .04 percent of the working memory cells would be active using a local representation, while in the distributed representation about 7.5 percent are in use. Our second reason for preferring a distributed representation is that a direct tie between individual neurons and symbolic structures is physiologically implausible; it is reminiscent of the yellow volkswagen cell idea.

Using a distributed representation based on coarse coding we are able to cover the entire space of 15,625 triples with just 2000 cells. Each cell has a "receptive field" of 6^3 or 216 triples, defined by the cross product of six randomly chosen symbols in each of the three positions of a sequence. For example, the cell described in figure 2 has the triples (C B R) and (F A A) in its receptive field, along with 214 other triples. The 2000 cells have slightly overlapping receptive fields: the average number of triples in the intersection of two cells' receptive fields is less than one. Yet in another sense there is a large degree of overlap, because each of the 15,625 possible triples falls within the receptive field of, on average, 28 different cells.

3.2. Storing Triples in Working Memory

"Storing" a triple in working memory using a coarse coded representation means turning on all the WM cells in whose receptive field it falls. On average this is about 28 cells; the number varies from one triple to another due to the random distribution of receptive fields. To test if a particular triple is present in working memory, we can check the fraction of active cells among those that can receive it. If this fraction is close to 1.0, we may assume the triple is present. For example, let us store the triple (F A A) in working memory. To do so we will turn on the neuron described in figure 2, since (F A A) falls within its receptive field; we will also turn on about 27 other neurons.

Notice that (C B R) falls within the receptive field of the cell in figure 2. The total number of receptors common to two unrelated triples is small; the average number is slightly less than one. While 28 out of 28 (F A A) cells are active, only 1 out of about 28 (C B R) cells will be active. Thus we can state with a high degree of confidence that (F A A) is present in working memory but (C B R) is not.

C	A	A
F	B	D
M	H	J
P	K	M
S	S	P
W	Z	R

Figure 2: The receptive field table of a WM cell.

3.3. Properties of Coarse Coding

Coarse coded memory representations have several interesting features. One is immunity to noise. If we store some triples in WM, then turn a few cells on or off at random, the perceived contents of WM will not change. This is very important because we have allowed some overlap in the representation of triples: as production rules add and delete certain triples from working memory, the overlap will gradually affect the representation of other triples stored there. But because the overlap is small (due to small receptive field size) and the system is immune to small amounts of noise, the contents of WM are reasonably persistent.

Another interesting feature of the distributed representation is that it gives a gradual degradation of WM performance as the number of elements increases. Each triple added to WM increases the number of active cells, and therefore increases the overlap with triples that have not been explicitly added. As WM fills up, the fraction of active cells for triples that are "close" to those that have been stored approaches 1.0, and the dividing line between present and absent triples blurs. If many closely related triples are stored, such as (F A A), (F A B), (F A C), etc., then the system may exhibit local blurring, where it can't tell whether (F A X) is present or not, but it is certain that (G K Q) is absent.

3.4. Clause Cells

Each production rule contains exactly two clauses on the left hand side, where a clause is a specification of a triple. Since there are two clauses, each rule must match a pair of WM triples. Working memory holds half a dozen triples on average. The clause cells in C1 and C2 provide a way to pull out specific working memory triples so they can be matched against the clauses in the production rules. Michael Mozer of UCSD has independently invented a device similar to clause spaces, which he calls "pull out networks," to allow a perception system to attend to specific objects in a scene (Mozer, 1984).

There are 2000 cells in C1 space in one-to-one correspondence with the WM cells; the same is true for C2 space. Each WM cell has an excitatory connection to its corresponding C1 and C2 cells. Thus, whenever a WM cell comes on, it tends to turn on the corresponding cells in the C1 and C2 spaces. However, clause cells have a mutually inhibitory influence within their own space which is designed to limit the number of active clause cells to about 28 per space, *i.e.* just enough to represent one triple. The number of active cells in WM space is not regulated. Thus, while WM may hold a half dozen or more triples, when the network settles C1 and C2 spaces will ideally hold just one triple each that has been selected out of WM space.

It might appear possible for the C1 and C2 spaces to settle into states representing triples that bear no relation at all to the contents of working memory, but instead simply contain 28 active neurons. This is a highly unlikely occurrence because a randomly chosen activation pattern in clause cell space will receive very little support from the Rule and Bind cells. The system's thresholds and bias levels have been chosen so that a clause cell cannot remain active unless it receives support from a reasonable number of both Rule and Bind cells as well as its corresponding WM cell.

4. Representation of Rules

Each production rule is represented by a population of 40 Rule cells. Let us begin with our first production system interpreter, where the rules contain no variables, as in Rule-1 above. The left hand side of this rule references the triples (F A A) and (F B B). Each Rule cell that contributes to the representation of Rule-1 receives input from a random subset of the (F A A) cells in the C1 population, and an equal number of randomly chosen (F B B) cells in the C2 population. If a sufficiently large number of C1 and C2 cells are active, indicating that the triples (F A A) and (F B B) are present in working memory, the Rule cell will also become active.

The 40 cells representing one production rule form a clique. Each cell in the clique provides a slight excitatory stimulus to the other cells in the clique, and a slight inhibitory stimulus to the Rule cells in other cliques. Thus, the Rule space is organized as a "winner take all" network (Feldman and Ballard, 1982); when the network settles, all the cells in one clique will be active and all the remaining cells will be inactive. This is how the system decides which rule to fire.

One reason for implementing rules as collections of cells rather than as single Rule cells is that it allows for a graded response. If, during the settling phase, there is a weak match between one rule and working memory, this will be indicated by only some of the the corresponding Rule cells being active. If another rule matches more strongly, more of the cells in its clique will be active, and they will eventually inhibit the cells in the other cliques.

Another reason for implementing rules with multiple cells is that it frees any one cell from having to represent the entire pattern associated with a production rule's left hand side. Instead, each Rule cell has just a small amount of information; only the clique as a group has the complete representation for the rule. This is a more plausible organization than one in which each rule is represented by a single cell, since it allows us to limit the number of connections each Rule cell must make to other cells.

5. Settling

Hopfield (1982) has shown that the state of a neural network with symmetric connections between units can be usefully described using the following energy measure, where s denotes the state (0 or 1) of the *i*th neuron, θ_i denotes the threshold of the *i*th neuron, and w_{ij} denotes the weight of the connection between the *i*th and *j*th neurons:

$$E = \sum_i s_i\theta_i - \sum_{i<j} s_i s_j w_{ij}$$

If neurons change state asynchronously and there is no transmission delay across connections, such networks are guaranteed to settle into a minimum energy state from any starting state. This analogy to physical systems is the basis of the Boltzmann Machine architecture (Fahlman *et al.*, 1983; Ackley *et al.*, 1985), but it is also important for non-Boltzmann neuron simulators such as the one discussed here. By designing the weights in our production system interpreter so that a successful rule match corresponds to a low energy state, we can match production rules against working memory by starting the network in a high energy state and allowing it to settle into an energy minimum. This is not a foolproof match technique; some problems with it will be discussed later.

6. Rule Firing

The right hand side of a rule consists of a set of triples to add to working memory and a set to delete from it. A rule can add triples by exciting the WM cells that receive those triples, and it can delete triples by inhibiting those same WM cells. Thus, the right hand side of a rule specifies two populations of WM cells: those to be excited and those to be inhibited. The 40 Rule cells that represent a rule each make connections (of the appropriate type, either excitatory or inhibitory) to a random subset of the total population of cells the rule is to affect. However, these connections are gated so that the Rule cells can only influence the WM cells during rule firing, rather than all the time, and the WM cells cannot influence the rule cells through symmetric connections. Although this would appear to violate Hopfield's conditions, we can show that during each settling phase the network is equivalent to another network that does not violate these conditions, and thus its behavior during a settling can be understood in terms of energy minimization even though the whole sequence of settlings cannot.

Once the network of C1, C2, and Rule cells has settled into a stable state indicating a match, it is a simple matter to fire the right hand side of the rule that matched. This is the rule whose clique of Rule cells is active. All we need do is open the gate on the connections between Rule space and WM space. Each active Rule cell will supply a small amount of inhibition or excitation to certain WM cells. If a cell receives enough of these inputs, its state will be changed. Once the gate is closed, WM cells retain their most recent state until the gate is opened again at the next rule firing.

Consider the case where Rule-1 has matched successfully, and it is now time to fire its right hand side. Each Rule cell will supply excitatory inputs to some of the WM cells that receive the triple (G A B), and inhibitory inputs to some of the WM cells that receive (F A A) or (F B B). To guard against a stray Rule cell upsetting the contents of working memory, the weights and thresholds are set so that the concerted action of several Rule cells is required to change the state of a WM cell in either direction. In other words, WM cells exhibit hysteresis.

The distributed nature of the rule representation means no single Rule cell contains a complete representation of a production rule; a rule's successful matching and firing does not critically depend on the behavior of any single cell or small group of cells; and during rule firing a few Rule cells can be turned on or off at random without effecting the updating of working memory at all.

7. Variable Binding

Let us now consider rules where a variable appears on the left hand side. In the system as it is currently implemented, the variable must appear in the first position of each triple. Rule-2 , whose left hand side contains (=x A B) and (=x C D), is an example. This rule can match pairs of triples such as (F A B) and (F C D), but it cannot match the pair (F A B) and (G C D) because the symbol in the first position of each triple is different.

To represent the binding of the variable =x we use a device called Bind cells. These are similar to the mapping units used for object recognition in (Hinton, 1981). Since there are 25 separate symbols in our alphabet, the variable =x can have 25 possible values. We represent each possible value by a population of 40 Bind cells, so there are 40 cells for the symbol A, 40 for the symbol B, and so on. Bind cells receive input from cells in both C1 and C2 space, and also influence the cells in those spaces. For example, each F cell receives input from a random subset of the C1 and C2 cells that have an F in the first column of their receptive field tables. Each group of 40 Bind cells forms a clique; every cell in a clique excites its neighbors slightly, and also slightly inhibits the cells in the other cliques. Thus, Bind space is another winner-take-all network.

Suppose that WM contained the triples (F A B), (F C D), and a few other random triples such as (G Q K). Suppose Rule-2 was present in the network's long term production memory (*i.e.* in the connections of the appropriate rule cells.) Then as settling progressed C1 space would settle into the representation of (F A B) and C2 space would settle into the representation of (F C D). Each F Bind cell would be getting excitation from several cells in each of C1 and C2 space, so the F Bind cells would become the active clique. Also, as the F bind cells become active, they tend to support C1 and C2 cells representing triples that begin with F, thereby strengthening the representation of (F A B) and (F C D) in their respective clause spaces.

One key difference between rules with variables and rules without is in the receptive field size of the Rule cells. There are about $(6/25)^3 \times 2000 = 28$ cells that can receive (F A B), but about $(6/25)^2 \times 2000 = 115$ cells that can receive (=x A B) for any value of =x. So the Rule cells must be given larger receptive fields and different connection strengths and thresholds in order to cover the larger number of clause cells matching a triple with a variable in it. A production system interpreter capable of accepting mixed rule types (*i.e.* some with variables, some without) would be a logical extension to our second system.

The settling process by which rule matching is accomplished with variable binding is similar to what was described earlier, except that now C1 and C2 cells are influenced by both Rule cells and Bind cells, acting independently. However, the two populations of cells tend to work together to force the C1 and C2 cells into representing triples that give a legal rule match.

8. Performance

Both production systems have run successfully on small test cases (sets of about six rules operating on a working memory holding two to six elements at a time.) In one test, which involved a finite state machine cycling through a series of six distinctive WM configurations, the system ran (overnight) through more than a thousand rule firings with no evidence of memory deterioration or other difficulty. A similar test using rules that involve variable binding gave equally encouraging results.

However, we have also found situations that cause problems for the settling algorithm used in rule matching. A trivial case is one where no rule successfully matches working memory; the system will still settle into some sort of local energy minimum, since it must do so. However, it may be possible to detect this no-match condition if it turns out that all good matches have "deep" minima and unsuccessful matches have only shallow minima. In preliminary experiments using two sample production systems, this has in fact been the case. Our interpreter was able to reject faulty matches by checking whether the final settling energy of the system exceeded a given threshold. In that case, rather than going on to the firing phase it throws away the match and runs a new settling phase.

If there is more than one possible successful match, the two possibilities may interfere with each other. Since the Rule cells and Binding cells compete independently; the state the system finally settles into may have two active cliques in Rule and/or Bind space, or there might be no active cliques left in one of the spaces. We have chosen to make the simplifying assumption that exactly one rule (with one binding) will be firable during each recognize-act cycle. However, it turns out that this assumption does not eliminate the possibility of interference among Rule or Bind cells.

Consider a simple system of five production rules with no variable binding. The first four rules all reference the triple (A A A) which is present in working memory, and some other triples which are not present. The fifth rule references the triples (B B B) and (C C C), both of which are present. Working memory also contains some additional random triples. During settling, the C1 cells corresponding to the representation for (A A A) will get support from four cliques of Rule cells, although the Rule cells will themselves be only weakly supported because they can get support for their C1 clause but not their C2 clauses. On the other hand, the C1 cells corresponding to (B B B) will be supported only by one clique, since only one rule references that triple, and similarly for the C2 cells representing (C C C). In this case, although only one of the five rules can be fired correctly, the system may still settle on the wrong rule due to the combined influence of the unsuccessful rules, or it may settle into a minimum that does not not represent a successful match at all.

Obviously, when variable binding is permitted in rules, the potential for unsuccessful settling is increased. One way around such problems might be to use simulated annealing (Kirkpatrick, 1983) as the search technique rather than doing a straight gradient descent in energy space. Simulated annealing is a way to avoid getting stuck in local minima, so if there is a good match to be found, we can usually find it. We would then be adopting the Boltzmann approach (Ackley *et al.*, 1985), which is computationally more expensive to simulate than gradient descent, but a much more effective search technique. We are pursuing this idea in our next generation production system interpreter.

9. Conclusions

We have described an implementation of production systems on a neural network architecture in which two common symbolic reasoning operations, pattern matching and variable binding, were performed using distributed representations. The work demonstrates that connectionist architectures are not limited to solving low-level vision problems or implementing associative memory schemes; they can be programmable symbol processors. The success of our production system implementation goes some way towards answering a common criticism of connectionist theories: that they aren't powerful enough to do symbolic reasoning.

Our results also serve as the beginnings of a theory of symbolic representation in the brain. While the details of our model are not physiologically correct, we have nonetheless made progress by showing how distributed symbolic representations, which are physiologically plausible, can be manipulated effectively.

The brain is built from painfully slow and unreliable components: neurons, which fire less than once per millisecond, are susceptible to fatigue, and die off regularly. The only way the brain can succeed as a symbol processor is by exploiting massive parallelism using organizational principles that remain unknown for the present. By exploring the problem of computing with distributed representations, computer scientists may eventually uncover some of these principles.

Acknowledgements

This research was supported by a grant from the System Development Foundation. We thank Scott Fahlman, Jay McClelland, David Rumelhart, and Terry Sejnowski for helpful discussions.

References

Ackley, D. H., Hinton, G. E., & Sejnowski, T. J. A learning algorithm for Boltzmann machines. *Cognitive Science*, 1985, 9, 147-169.

Ballard, D. H., Hinton G. E. & Sejnowski, T. J. Parallel visual computation. *Nature*, 1983, 306, 21-26.

Ballard, D. J. & Hayes, P. J. Parallel logical inference. *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*. Boulder, Colorado. June, 1984.

Barrow, H. G. & Tenenbaum, J. M. Recovering intrinsic scene aracteristics from images. In A. Hanson and E. Riseman s.), Computer Vision Systems. New York: Academic Press, 8.

Cottrell, G. W. A model of lexical access of ambiguous words. In *Proceedings of the National Conference on Artificial Intelligence*. Austin, Texas: August 1984.

Fahlman, S. E. Hinton, G. E. & Sejnowski, T. J. Massively parallel architectures for AI: Netl, Thistle, and Boltzmann Machines. In *Proceedings of the National Conference on Artificial Intelligence*. Washington D.C.: August 1983.

Feldman, J. A. & Ballard, D. H. Connectionist models and their properties. *Cognitive Science*, 1982, 6, 205-254.

Hinton, G. E. A parallel computation that assigns canonical object-based frames of reference. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vol 2, 683-685. Vancouver BC, Canada. August 1981.

Hinton, G. E. Shape representation in parallel systems. In *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vol 2, 1088-1096. Vancouver BC, Canada. August 1981.

Hinton, G. E. & Anderson, J. A. (Eds.) *Parallel Models of Associative Memory*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1981.

Hinton, G. E., McClelland, J. M. & Rumelhart, D. E. Distributed representations. In D. E. Rumelhart and J. L. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1. Cambridge, MA: Bradford Books, 1985.

Hopfield, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences USA*, 1982, 79, 2554-2558.

Kirkpatrick, S. Gelatt, C. D. & Vecchi, M. P. Optimization by simulated annealing. *Science*, 1983, 220, 671-680.

McClelland, J. L. & Rumelhart, D. E. An interactive activation model of the effect of context in perception: Part 1. An account of basic findings. *Psychological Review*, 1981, 88, 357-407.

Marr, D. & Poggio, T. A theory of human stereo vision. *Proceedings of the Royal Society of London B*, 204, 1979.

Minsky, M. & Papert, S. *Perceptrons*. Cambridge, Mass. MIT Press, 1969.

Mozer, M. C. The perception of multiple objects: a parallel, distributed processing approach. Unpublished thesis proposal, Institute for Cognitive Science, University of California, San Diego. La Jolla, CA: 1984.

Post, P. B. A lifelike model for associative relevance. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 271-280. Washington, DC. May, 1969.

Rumelhart, D. E. & McClelland , J. L. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* Volume 1. Cambridge, MA: Bradford Books, 1985.