Neuro-Soar: A Neural-Network Architecture for Goal-Oriented Behavior*

Bonghan Cho and **Paul S. Rosenbloom**
Computer Science Department
University of Southern California
Los Angeles, CA 90089
Email: bcho,rosenbloom@ISI.EDU

Charles P. Dolan
AI Center, Hughes Research Labs
3011 Malibu Canyon Rd.
Malibu, CA 90265
Email: cpd@aic.hrl.hac.com

Abstract

The ability to set and achieve a wide range of goals is one of the principal hallmarks of intelligence. The issue of goals, and of how they can be achieved, has been one of the major foci of Artificial Intelligence (AI), and the understanding of how to construct systems that can accomplish a wide range of goals has been one of the major breakthroughs provided by the study of symbolic processing systems in AI. Neural networks, however, have not shared this focus on the issue of goals to any significant extent. This article provides a progress report on an effort to incorporate such an ability into neural networks. The approach we have taken here is to implement a symbolic problem solver within a neural network; specifically we are creating Neuro-Soar, a neural-network reimplementation of the Soar architecture. Soar is particularly appropriate for this purpose because of its well-established goal-oriented abilities, and its mapping onto levels of human cognition — in particular, the ways in which it already either shares, or is compatible with, a number of key characteristics of neural networks.

Introduction

An ability to set and achieve a wide range of goals is one of the principal hallmarks of intelligence. The issue of goals, and of how they can be achieved, has been one of the major foci of Artificial Intelligence (AI), and the understanding of how to construct systems that can accomplish a wide range of goals has been one of the major breakthroughs provided by the study of symbolic processing system in AI. Neural networks have not shared this focus on the issue of goals to any significant extent. However, if neural networks are to be serious contenders as the basis for a complete intelligent system, they must eventually incorporate the ability to set and achieve the full range of goals faced by such systems.

This article provides a progress report on an effort to incorporate such an ability into neural networks. The approach we have taken here is to implement a symbolic problem solver within a neural network; specifically we are creating Neuro-Soar, a neural-network reimplementation of Soar — a symbolic architecture for intelligence that integrates basic mechanisms for problem solving, use of knowledge, learning, and perceptual-motor behavior [Laird *et al.*, 1987]. Soar[1] is particularly appropriate for this purpose because of its well-established goal-oriented abilities, and its mapping onto levels of human cognition [Newell, 1990] — in particular, the ways in which it already either shares, or is compatible with, a number of key characteristics of neural networks [Rosenbloom, 1989]. Work to date on Neuro-Soar covers the lowest three levels of Soar — memory accessing, decision making, and (problem space) operations. Together these capabilities are sufficient to enable goal-oriented behavior in a single problem space, but are not yet sufficient to enable higher-level search and learning behaviors. One of the key developments in accomplishing this is the implementation of a novel neural-network-based production system that allows multiple variables per production, and parallel matching and firing of multiple productions.

*This research was partially supported by a gift from the Artificial Intelligence Center of the Hughes Aircraft Company.

From Soar to Neuro-Soar

In Soar, all symbolic goal-oriented behavior is formulated as steps in problem spaces. A problem space is a uniform task representation that allows Soar to be applied to a wide range of goals. The problem space determines the set of states and operators that can be used during the processing to attain a goal. Goals, problem spaces, states, and operators exist as data structures in Soar's working memory — a transient declarative memory. Each goal defines a problem solving context which contains roles for a problem space, a state, and an operator.

Problem solving is driven by the acts of selecting problem spaces, states, and operators for the appropriate roles in the context; where each selection is accomplished via a two-phase decision cycle. First, during the *elaboration phase*, the description of the current situation (that is, the content of working memory) is elaborated with relevant information from long-term memory — an associative memory, constructed as a parallel production system. The rules contain variabilized conditions that are matched against the contents of working memory, and actions that add elements to working memory when all of the conditions are met. One important type of knowledge that may be added during the elaboration phase is *preferences*. There is a fixed

[1] In this article, we focus on Soar version 4.

language of preferences which is used to describe the acceptability and desirability of the alternatives being considered for the selection. There are two types of preferences, unary and binary. The former states an absolute preference to all others (e.g., "best"), while the latter states a preference between two objects (e.g., "better"). In the second phase, the *decision phase*, the preferences in working memory are interpreted by a fixed decision procedure. If the preferences uniquely specify an object to be selected for a role in a context, then a decision can be made, and the specified object becomes the current value of the role. However, if the preferences for a decision are either incomplete or inconsistent, then an impasse occurs. Soar automatically generates a subgoal for the task of resolving the impasse and creates a new problem solving context within which it can be resolved. Soar learns by acquiring new productions which summarize the processing that leads to results of subgoals, a process called chunking.

Soar has been used to implement goal-oriented systems in a wide variety of domains: immediate reasoning tasks, simple puzzles and games, classical expert system domains, robotic control tasks, and natural language understanding. It has also demonstrated the ability to utilize a wide variety of problem solving methods. However, despite all of this high-level symbolic capability, much of the architectural functionality described above can be mapped to corresponding neural network concepts (at least at a gross level). In particular, Soar's production memory is just a fine-grained parallel associative memory — though it does work with graph-structured objects and does allow multiple variables — and its decision procedure is just a method for locally integrating together multiple constraints on selection, as is generally done via activation combination in neural networks. The need to allow parallel matching and firing of productions over graph-structured objects with multiple variables adds some complexity to this mapping; however, in contrast to traditional production systems, a novel formulation — that restricts attributes to having at most one value at a time (referred to as the *unique-attribute* formulation) — has been developed for Soar that greatly simplifies implementation in neural networks by guaranteeing both linear-time match complexity and a maximum of one binding per variable [Tambe *et al.*, 1990].

In the remainder of this article we focus on how Soar's production system and decision procedure can be implemented in a neural network, and how these two capabilities can be combined to enable the selection and application of sequences of problem-space operators that solve simple blocks world problems. Impasse detection, subgoal generation (with the concomitant abilities of using multiple problem spaces and searching), and learning are left for another day.

Representing Problem Solving Knowledge

In problem solving behavior in a simple blocks world problem (Sussman's anomaly) three blocks are arranged on a table, and there are operators that allow blocks to be moved onto either other blocks or the table. In the first situation, three operators are suggested, and sufficient preferences are retrieved from long-term memory to allow the correct operator to be selected. Once it is selected, further memory retrieval effectively applies the operator

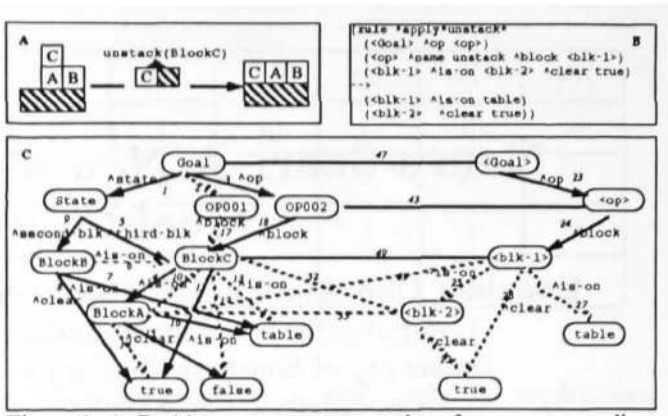

Figure 1. A. Problem space representation for operator application. B. Corresponding production rule. C. The graph view. One unit will be assigned to each number. Some edges for the working memory and bindings are not shown for clarity.

by modifying the problem solving state (moving block C to the table) Problem solving then continues by repeating the entire process.

Figure 1A focuses on a fragment of this overall behavior — the application of the selected operator to modify the state. In the remainder of this section we focus on how the knowledge is represented in Soar and the transformations that lead from this representation to the one used in Neuro-Soar. Figure 1B provides a *text view* of how this is represented in Soar (in a slightly simplified form) as a production. The conditions are prior to the arrow and the actions after. Symbols preceded by " ^ " and bounded by " <> " are attributes and variables, respectively.

To get from this text view of the information to the *neural view*, it is useful to go through an intermediary *graph view*, as shown in Figure 1C. In the graph view, each individual triple of an object, attribute, and value is represented by a labeled directed edge from the object to the value. In the working memory, a solid edge indicates a working memory element and a dashed edge indicates an element that potentially could be in working memory, but currently isn't. This distinction is not needed in the text view, but is crucial for the neural view. In production memory, there are two types of edges, one for the conditions (denoted with straight edges), and the other for actions (denoted with jagged edges). Solid and dashed edges denote matched and unmatched triples, respectively. The bindings are represented by undirected edges between the vertices of the working memory graph and the production memory graph. A solid edge indicates a binding of a production variable to an object in working memory, and a dashed edge indicates a potential binding.

A match between the working memory and a production occurs when there is an exact match between a subgraph of working memory and the production graph through a set of bindings[2]. For a successful match, there will be exactly one solid edge from each vertex in the production to some vertex in the working memory.

Production systems in the neural view are composed of relatively simple, neuron-like processing units ("units" for

[2]In contrast to Soar 4, preferences are not in working memory, instead they exist in preference memory, which cannot be matched by productions.

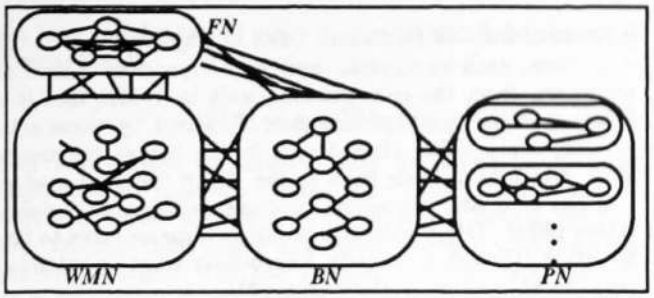

Figure 2. Network Architecture of Neuro-Soar.

short) and weighted connections between them. In the neural view, there is one unit for each edge in the graph view. The details are provided in the following section.

Representation in Neuro-Soar

Neuro-Soar is composed of four sub-networks and connections between them (see Figure 2). Three of the sub-networks correspond to the three components described in the previous section: the working memory network, the production network, and the binding network. The fourth sub-network is the preference network, which did not appear in the previous section because it is used in the selection of operators, but not in their application.

The role of the working memory network (WMN) is to represent the contents working memory as a combination of unit activations. In the neural view, each WMN unit corresponds to an edge in the graph view. WMN units are activated by production actions, and deactivated when there is no longer an activated path between them and the goal node. This deactivation process corresponds to a garbage collection process in the symbolic implementation of Soar.

The production network (PN) is actually a set of networks, one for each production. Each network can be activated individually, in parallel, by the active segments of WMN via the binding network. To create a PN for a particular production, condition and action units are created for the condition and action edges in the graph view (one unit per edge). Each production also needs a rule unit to detect the joint activations of all the condition units. Each condition unit is connected to the rule unit, and the threshold of the rule unit is simply the number of condition units. The rule unit activates the action units, which in turn activate working memory units.

The binding network (BN) is the vehicle by which conditions are matched to working memory, and by which actions have their effects on working memory. In so doing, it computes variable bindings between the PN and WMN, and ensures their consistency — each instance of the same variable in a production must bind to the same object, each object must bind to no more than one variable[3], and no more than one object can be bound to any variable at the same time. The basic binding units are obtained from edges in the graph view of binding. Each binding unit represents a particular instantiation of a variable of a production to a (potential) object in working memory.

Figure 3 shows how consistent bindings are computed.

[3]Soar actually enforces this restriction in learned rules, but not in programmed rules. In Neuro-Soar it is enforced for all rules.

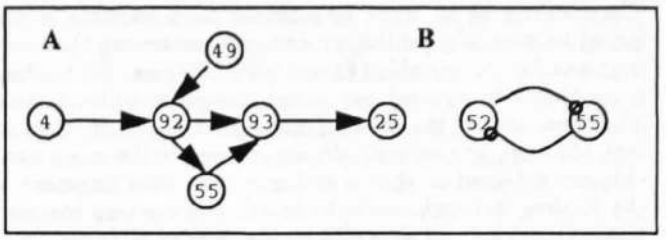

Figure 3. **A**. Matching mechanism in Neuro-Soar. Units 4 and 25 are WMN and PN units, respectively. Units 49 and 55 are binding units. Units 92 and 93 are instantiation units. **B**. Mutual inhibition among units whose bindings are contradictory to each other, through which consistent variable bindings are computed.

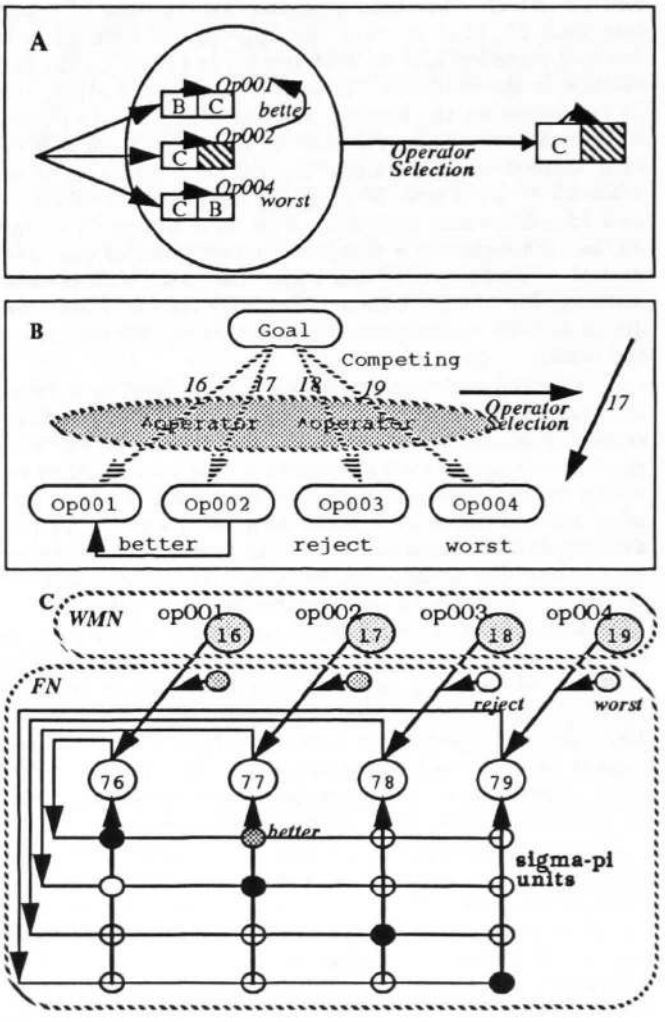

Figure 4. Neuro-Soar decision procedure. *A*. Problem space representation for operator selection. **B.** Corresponding graph view. **C.** Neural network implementation of corresponding preference information. A new layer of units (76,77,78,79) is introduced where mutual inhibitory connections exist by default. Preferences are stored in sigma-pi units (small units). A selection of unit 77 by the neural decision procedure will feed back to unit 17 in the WMN.

The binding of at most one object to a variable is enforced by mutually inhibitory connections among the binding units for the variable (Figure 3B). Likewise, the binding of an object to at most one variable is enforced by mutual inhibition among the binding units for the object. Consistent bindings across multiple occurences of the same variable are enforced as shown in Figure 3A. This fragment of the binding network exists to match the working memory triple (BlockA ^ on BlockB) to the condition (<blk-1> ^ on <blk-2>). As before the units in the figure are identified by their numbers: 4 is the working memory unit, 25 is the condition unit (in PN), 49 and 55 are binding units, and 92 and 93 are (partial) instantiation units. In the current situation, both unit 4 (representing that BlockA is on BlockB) and unit 49 (representing the binding of BlockA to <blk-1>) are active. This combination then activates unit 92, which represents a partial instantiation of condition unit 25; that is, that the appropriate element is in working memory, and its first symbol is bound to the first variable in the condition. To pursue the match further, unit 55 (representing the binding of BlockB to <blk-2>) needs to be activated. This unit receives activation from unit 92, so if there is no competition from other binding units for <blk-2> — or if unit 55 simply wins the competition — unit 55 will become activated. Unit 93 is analogous to unit 92, but it represents a complete instantiation of condition unit 25 — there would be one unit like 93 for each possible instantiation of condition unit 25. This then activates condition unit 25, thus successfully completing this segment of the match.

The overall match starts with an initial binding between the goal object in working memory and the first condition in each production. From there, the match ripples down the networks, with each new binding that is established enabling further matches to proceed (or be inhibited). When all of the conditions of a production are matched, the production unit becomes active, and in turn activates the action units. The action units then modify working memory activations through an inverted binding network.

In addition to the straightforward effect of action units on working memory, action units can also modify working memory indirectly by asserting preferences among units in WMN — this is how problem solving operators are selected. The role of the preference network (FN) is to reflect these current preferences (during the elaboration phase) and to select a unit out of the competing candidate units, based on their preferences (during the decision phase).

Figure 4A shows the operator selection episode, elaborated with the preferences that lead to the selection of Op002 (put Block C on the table). This operator is selected because it is *better* than Op001, and the other candidate (Op004) is *worst*. Figure 4C shows how this behavior is implemented as a neural decision procedure in the preference network. The FN is built on top of the WMN as an additional layer that stores and processes preferences (see units 76, 77, 78, 79 in Figure 4C)[4]. The FN is basically formed as winner-take-all network with a guide of preferences. Unary preferences are encoded as wieghts between the WMN units and the FN units, while binary preferences are encoded as weights between the competing units in FN. The use of Sigma-Pi units[Rumelhart *et al.*, 1986] is required for this (see small units in Figure 4C).

At first, each competing unit in FN receives a level of activation from the correponding unit in WMN that is a function of its unary preferences. Prodded by these activations, competition then occurs in the binary preference layer, which then feeds back to the WMN units. If a selection can be made, processing continues with another elaboration phase. Otherwise, the resulting impasse leads to termination (though ultimately Neuro-Soar must be extended perform subgoal generation instead).

The Neuro-Soar approach of encoding preferences as constraints on activation combination is not guaranteed to yield the same outcome as would be achieved through Soar's decision procedure, but it does capture the spirit of their meaning. The extent to which the Neuro-Soar scheme is actually better or worse than the symbolic approach is still to be determined.

Experiments and Analysis

So far, Neuro-Soar has been applied in two domains: the blocks world and the tower of Hanoi. In this section, we focus on results from the blocks world. The implementation consists of 8 productions. Two test problems have been run — Sussman's anomaly and a simpler problem of unstacking a tower of three blocks. In both cases Neuro-Soar solved the problem by replicating the sequence of operator selections and applications that were produced by Soar when it was given the same knowledge.

Since Neuro-Soar employs a local representation scheme, a sufficient number of units in working memory must be preallocated. This number is dependent on the number of objects in working memory and their attributes. The number of units in PN is dependent on the number of conditions and actions in the productions. Likewise, the number of units in the binding network is related to the number of objects in working memory and the productions. Only a small number of units are required in FN since they are built on top of a small part of working memory. The blocks world implementation requires 135 WMN units, 119 PN units (of which there are 8 production units, 48 condition units, and 63 action units), 2148 BN units (of which there are 185 binding units and 1963 instantiation units), and 10 FN units. More generally, the worst space complexity of a task implemented in Neuro-Soar is as follows. Let $nUNIT_{netwrok}$ be the number of units in network *network*.

$$nUNIT_{PN} = O(nTrp)$$

$$nUNIT_{WMN} = O(\sum_{attr} nObjHas_{attr} * nObjOf_{attr})$$

$$nUNIT_{BN} = O(nOBJ_{WMN} * nOBJ_{PN} + \sum_{attr} nWMN_{attr} * nPN_{attr})$$

$$nUNIT_{FN} = O(nOBJ_{WMN})$$

where $nTrp$ is the number of triples in productions; $nObjHas_{attr}$ and $nObjOf$ are the number of objects of working memory that can have attribute *attr* and can be the value of *attr*, respectively; $nOBJ_{network}$ is the number of objects in *network*; and $nWMN_{attr}$ and nPN_{attr} are the number of units which have attribute *attr*.

The time complexity for matching a production in Neuro-Soar is on the order of the maximum depth of the graph representation of the production. The depth of the graph representation is approximately $log(nTrp)$. The matching process is relatively fast and efficient in terms of neurosimulation cycles, not only because the rules are fired in parallel, but also because the valid partial matches

[4] In reality, there is also one additional layer that handles *indifferent* preference.

are kept (analogous to state saving in the Rete algorithm [Forgy, 1982]).

Related Work

The two classes of related work to which it is most important to compare Neuro-Soar (other than Soar itself) are: other attempts at constructing neural production systems, and other attempts at constructing neural problem solvers. On the former, two previous systems have implemented production systems using neural networks, the Distributed Connectionist Production System by Touretzky and Hinton[Touretzky and Hinton, 1988] and the Tensor Product Production System by Dolan and Smolensky[Dolan and Smolensky, 1988], both of which used distributed representations. Neuro-Soar, though simpler in that it uses a local representation, goes significantly beyond these two systems in allowing parallel matching and execution of productions, (unique-attribute) graph-structured representations, and multiple variables per production.

Ajjanagadde and Shastri[Ajjanagadde and Shastri, 1990] have developed a connectionist system that represents knowledge as rules and facts with multi-place predicates. A multi-phased clock is proposed as a solution to the variable binding problem. The system produces answers, when given 'yes/no' questions, based on the knowledge. However, the system lacks a capability to create and alter the content of its temporary knowledge so that situation dependent firing of rules (or planning) is not possible.

This is by no means the first effort to construct a neural-network-based problem solver (see, for example, [Sutton and Barto, 1981, Anderson, 1989]). However, the current effort differs from most of the earlier approaches in adding representational and structural complexity — specifically, graph-structured objects and the decomposition of the overall memory into four specialized subnetworks — in service of domain generality. The hope is that this will allow Neuro-Soar to be extended up to Soar's full level of capability, without ultimately sacrificing learnability or neural plausibility.

Conclusions and Future Work

Work to date on Neuro-Soar has demonstrated that it is possible to reproduce key aspects of Soar's general goal-oriented ability even when the implementation technology is shifted from "symbolic" to "neural network". In particular, we have demonstrated a domain-independent approach to knowledge-driven operator selection and application. To accomplish this, we have developed a preference-based neural-network decision procedure, and a novel neural-network production system that allows parallel matching and firing of rules, graph structured objects, and multiple variables per rule.

However, much remains to be done in three general areas. The first area is the completion of Neuro-Soar with respect to Soar's functionality. This primarily involves the incorporation of impasse detection and subgoaling, use of multiple problem spaces, and learning. The second area is the loosening up of the design so that it less slavishly follows how things are currently done in Soar, and more takes advantage of the strengths of neural networks. We have begun with the conservative approach of trying to model the existing Soar architecture as closely as possible; however, we eventually want to get to where we can ask whether the specification of Soar ought to be altered because of the advantages accruable from using neural networks. For example, what would happen if we were to shift to a distributed representation for memory, based perhaps on a tensor-product formulation [Smolensky, 1990], or if we were to shift from a neural-network analogue of chunking to a more typically neural learning algorithm. Would such changes result in a more robust system that can learn more effectively? The third area is to see whether Neuro-Soar provides leverage in interfacing Soar to external environments. Would encoding the required signal-to-symbol transducers in neural-networks that are directly connected to Neuro-Soar provide a more seamless interface than is now provided by the use of Lisp code for the transducers?

References

[Ajjanagadde and Shastri, 1990] V. Ajjanagadde and L. Shastri. An optimally efficient limited inference system. In *Proceedings of AAAI 90*, pages 563–570, 1990.

[Anderson, 1989] C. W. Anderson. Tower of hanoi with connectionist networks: Learning new features. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 345–349, 1989.

[Dolan and Smolensky, 1988] C. P. Dolan and P. Smolensky. Implementing a connectionist production system using tensor products. Technical Report UCLA-AI-88-15, University of California, Los Angeles, 1988.

[Forgy, 1982] C. L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[Laird *et al.*, 1987] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[Newell, 1990] A. Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA, 1990.

[Rosenbloom, 1989] P. S. Rosenbloom. A symbolic goal-oriented perspective on connectionism and soar. In R. Pfeifer, editor, *Connectionism in Perspective*. North-Holland, 1989.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and J. L. McClelland. A general framework for parallel distirbuted processing. In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group, editors, *Parallel Distributed Processing, Volume 1: Foundations*, pages 45–76. Bradford Books/MIT Press, 1986.

[Smolensky, 1990] P. Smolensky. Tensor product variable binding and the representation of symbolic strucutre in connectionist systems. *Artificial Intelligence*, 46:159–216, 1990.

[Sutton and Barto, 1981] R. S. Sutton and A. G. Barto. An adaptive network that constructs and uses an internal model of its world. *Cognition and Brain Theory*, 4:217–246, 1981.

[Tambe *et al.*, 1990] M. Tambe, A. Newell, and P. S. Rosenbloom. The problem of expensive chunks and its solution by restricting expressiveness. *Machine Learning*, 5:299–348, 1990.

[Touretzky and Hinton, 1988] D. Touretzky and G. E. Hinton. A distributed connectionist production system. *Cognitive Science*, 12:423–466, 1988.