RAMBOT: A CONNECTIONIST EXPERT SYSTEM THAT LEARNS BY EXAMPLE

Michael C. Mozer

August 1986

ICS Report 8610

Institute for Cognitive Science
University of California, San Diego
La Jolla, California 92093

I owe an eternal debt of gratitude to Paul Munro, who christened the system RAMBOT and even managed to make RAMBOT into an acronym on the somewhat generic phrase "Restructuring Associative Memory Based On Training." I also wish to thank Paul Smolensky and Dave Rumelhart for their thoughtful comments. The robots program available at UCSD was written by Allan R. Black of Strathclyde University, and was modified by Stephen J. Muir at Lancaster University.

This research was supported by an IBM Graduate Fellowship, a grant from the System Development Foundation, and the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, Contract No. N00014-85-K-0450, Contract Authority Identification Number, NR 667-548. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the sponsoring agencies. Approved for public release; distribution unlimited. Reproduction in whole or in part is permitted for any purpose of the United States Government. Requests for reprints should be sent to Michael C. Mozer, Institute for Cognitive Science, C-015; University of California, San Diego; La Jolla, CA 92093.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ICS 8610	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Institute for Cognitive Science University of California, San Diego 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Personnel & Training Research Programs Office of Naval Research (Code 1142PT)
6c. ADDRESS *(City, State, and ZIP Code)* C-015 La Jolla, CA 92093	7b. ADDRESS *(City, State, and ZIP Code)* 800 North Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85-K-0450
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61153N	RR04206	RR04206-0A	NR 667-548

11. TITLE *(Include Security Classification)*
RAMBOT: A Connectionist Expert System That Learns by Example

12. PERSONAL AUTHOR(S)
Michael C. Mozer

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Technical	FROM 85 Oct TO 86 Apr	1986 August	15

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
05	10	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Connectionism; parallel distributed processing; learning by observation; artificial intelligence; expert systems; game playing

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Expert systems seem to be quite the rage in artificial intelligence, but getting expert knowledge into these systems is a difficult problem. One solution would be to endow the systems with powerful learning procedures which could discover appropriate behaviors by observing an expert in action. A promising source of such learning procedures can be found in recent work on connectionist networks, that is, massively parallel networks of simple processing elements. In this paper, I discuss a connectionist expert system that learns to play a simple video game by observing a human player. The game, Robots, is played on a two-dimensional board containing the player and a number of computer-controlled robots. The object of the game is for the player to move around the board in a manner that will force all of the robots to collide with one another before any robot is able to catch the player. The connectionist system learns to associate observed situations on the board with observed moves. It is capable not only of replicating the performance of the human player, but of learning generalizations that apply to novel situations.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Harold Hawkins	22b. TELEPHONE *(Include Area Code)* (202) 696-4323 22c. OFFICE SYMBOL ONR 1142PT

DD FORM 1473, 84 MAR
83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

Contents

RAMBOT: A Connectionist Expert System That Learns by Example

MICHAEL C. MOZER

INTRODUCTION

Expert systems are prominent among the successes of artificial intelligence. In fact, expert systems have become so popular that almost any program, if billed as an "expert" system, gains instant notariety. It's not always clear what is or is not an expert system, but the most interesting systems seem to operate in domains where the knowledge involved cannot be expressed in concise algorithms (Charniak & McDermott, 1985). Consequently, the most difficult task in building these systems is encoding the knowledge base (Duda & Shortliffe, 1983). Experts are often not as much help as one would like, because it is hard for experts to specify exactly what it is they're doing that makes them experts.

It would be desireable if expert systems could observe an expert in action and then discover rules of the domain based on their observations. This would allow the experts to do what they do best—perform—rather than what they do poorly—explain their own behavior. Of course, discovering the rules of any domain based on observation is a difficult task and requires powerful learning procedures. One promising source of such learning procedures can be found in the recent work on learning in multilayered connectionist networks (Ackley, Hinton, & Sejnowski, 1985; Barto & Anandan, 1985; Rumelhart, Hinton, & Williams, 1986). These networks have the ability to learn arbitrary associations from a set of known variables to a set of target variables or actions, say, from possible symptoms of a disease to possible treatments. More importantly, the networks are able to generalize from a set of examples to the broader class of situations they may be confronted with. While they generally do not discover explicit, psychologically real rules of the sort that most expert systems use, the behavior of these networks appears "rule governed" (Anderson & Hinton, 1981; Rumelhart & McClelland, 1986).

In this paper, I report on my initial efforts at constructing a connectionist expert system that learns to play a simple computer game by observing a human player. I begin by discussing some relevant properties of connectionist networks. Next, I explain the rules of the computer game, called *robots*, and present some strategy. I then describe my connectionist system, *RAMBOT*, which learns to play the robots game. Finally, I look beyond RAMBOT to consider the applicability of connectionist techniques to the design of expert systems in other domains.

CONNECTIONIST (PDP) SYSTEMS

Connectionist, or parallel distributed processing (PDP), systems are networks of simple processing elements that operate in parallel. The typical processing element has a large number of input lines and a single output line. The output line conveys a scalar value, called the *activation level*, and is generally a function of the weighted sum of the input lines. The output of a unit serves as input to other units or as an output of the system. Similarly, the inputs to a unit are received from other units or may be provided as input to the system.

A typical connectionist network architecture is shown in Figure 1. The network has three layers: an input layer, an intermediate layer, and an output layer. The input units are turned on by an external source, the input units then activate the intermediate units, and the intermediate units in turn activate the output units. This type of network implements an associative memory: an input activity pattern is mapped into an output activity pattern. Another way of conceptualizing the network is to imagine that each input unit represents some feature of an external environment and each output unit represents a possible action that could be taken. In this case, the network performs a stimulus-response mapping. For instance, the input units could signify the political climate of the world, the output units the possible actions that a nation might take (e.g., launch a nuclear attack, bomb innocent children, and so forth).

Learning in this network involves adjusting the strengths of connection, or weights, between units to implement the desired mapping. Until recently, the only known weight-adjusting algorithms were for two-layered networks, that is, networks with direct input/output connections, which are unable to learn many sorts of mappings. However, Barto and Anandan (1985), Ackley, Hinton and Sejnowski (1985), and Rumelhart, Hinton, and Williams (1986) have recently developed learning algorithms for multilayered networks like the one shown in Figure 1. I have been working with the *back propagation* algorithm of Rumelhart et al.

Using back propagation, a network can be trained to associate a set of paired input/output patterns. This training consists of two phases. In the activation phase, an input pattern is presented and is allowed to flow through the layered network to produce an output pattern. This output pattern is then compared with the target output pattern (the output that is to be associated with the given input) and a measure of discrepancy or error is computed. In the back propagation phase, the error is passed backwards through the network so that each unit has an indication of its contribution to the error. The back propagation algorithm implements gradient descent in the error measure; that is, it specifies a change in the weights that is guaranteed to decrease the error.

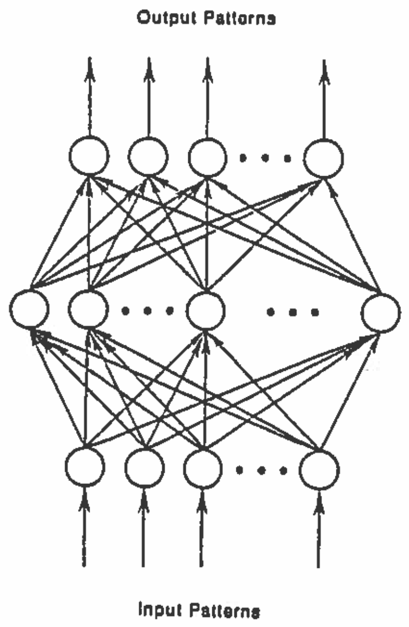


FIGURE 1. Typical connectionist network architecture.

Following learning, the network can perform arbitrary mappings between input and output patterns. More importantly, it is capable of generalization: If novel input patterns are presented, the network produces output patterns that appear reasonable in terms of the learned associations. To better understand the nature of such generalizations, consider the simplified case of a network that behaves linearly. Suppose this linear network has learned to associate input pattern A with output pattern A′ and B with B′. Then, presenting a pattern that is halfway between A and B will result in an output that is halfway between A′ and B′. Thus, generalization in this case consists of linear interpolation and extrapolation of learned patterns.

More realistically, multilayered networks require nonlinearities to achieve interesting sorts of behavior; these nonlinearities complicate the generalization issue. With the sorts of nonlinearities that are typical of back-propagation networks, it is still true that an input pattern composed of a mixture of A and B will produce an output pattern composed of a mixture of A′ and B′, though only if A and B are *sufficiently similar*. Unfortunately, "sufficiently similar" is difficult to define. A more sensible way of looking at generalization in a network like the one shown in Figure 1 is as follows. Think of the intermediate layer as performing a recoding of the input layer; that is, the intermediate layer constructs an *internal representation* of the inputs, one that is useful for solving the problem at hand. Generalization is then determined by the similarity among internal representations, not similarity among the actual input patterns. Thus, the response to a novel input pattern is similar to the response to known input patterns whose internal representations are similar to that of the novel pattern. Further, if the output units are linear or semilinear (see Rumelhart et al., 1986), the network performs the sort of interpolation and extrapolation described above, except it uses the internal representations of A and B, rather than A and B themselves.

The advantage of generalization should be obvious: the system needn't be trained on every point in the input space. If an unfamiliar input is presented, the system automatically determines its similarity to known inputs and produces a response based on this similarity. In contrast, many systems with explicit rules do not perform well in unfamiliar situations. Often, a missing or overspecified rule will cause the system to grind to a halt.

ROBOTS—THE GAME

I now return to the particular problem I've been working on: teaching a connectionist network to play the game robots.

The game is played on a CRT screen. The version I've worked with uses a 20×20 cell board. A sample board is shown in Figure 2A. The player is represented by an "I" and takes up one cell on the board. There are a varying number of robots, each represented by an equal sign. At the start of the game, the robots are placed on the board at random. On each turn, the player can move to an adjacent cell, remain at the current location, *teleport*, or *wait*. Teleport means that the player is lifted from the current location to a random location on the board; wait means that the player stays at the current location for the remainder of the game. The utility of these commands will be explained shortly.

After the player moves, each of the robots is allowed to move to an adjacent cell. The robots follow a simple algorithm. They march directly towards the player:

$$\mathit{robot\_delta\_x} = \operatorname{sign}(\mathit{player\_x} - \mathit{robot\_x}) \text{ and}$$
$$\mathit{robot\_delta\_y} = \operatorname{sign}(\mathit{player\_y} - \mathit{robot\_y}).$$

If two robots land in the same cell, they collide and are replaced by a *junk heap*. Junk heaps are inert and harmless to the player. The player must walk around junk heaps, but if robots collide with a junk heap, they are destroyed and become part of the heap. Figure 2B shows the game state one move after Figure 2A, where the player has moved left. As one can see, the two robots nearest to the player have collided and formed a junk heap, represented by an "@."

```
A  ----------------------      B  ----------------------
   |....................|         |....................|
   |..................=.|         |....................|
   |..=.................|         |.................=..|
   |....................|         |...=................|
   |....................|         |....................|
   |......=.............|         |....................|
   |....................|         |.......=............|
   |....................|         |....................|
   |.......=.......=..I.|         |........=.......@I..|
   |...............=....|         |....................|
   |....................|         |....................|
   |....................|         |....................|
   |....................|         |....................|
   |....................|         |.........=..........|
   |........=...........|         |..=.................|
   |.=..................|         |...........=.....=..|
   |..........=.....=...|         |....................|
   |....................|         |....................|
   |....................|         |....................|
   |....................|         |....................|
   ----------------------         ----------------------
```

FIGURE 2. A sample board before the player's move (A), and the board following one move by player and robots (B). The player moved to the left.

The player is forbidden from moving off the board, moving onto a cell with a junk heap or a robot, or moving adjacent to a robot. The player can die in one of three ways: by teleporting onto a robot, by teleporting onto a cell adjacent to a robot (in which case the robot moves over and crushes him), or by using the wait command when there is a robot directly in the line of sight (in which case the robot marches to the player and crushes him).

The object of the game is to kill off the robots by forcing them to run into each other or into junk heaps. When this happens, the game restarts at a higher level of difficulty, meaning that there are more robots on the board. The game begins at level 1 with 10 robots, and the number of robots increases to nearly 200 by level 9.

Tricks of the Game

Hiding behind junk heaps. If the player hides to one side of a junk heap and robots are approaching from the other side, the robots will march into the heap. For instance, the robot on the same row as the player in Figure 2B will eventually hit the junk heap. If in fact all robots are on the opposite side of the heap, the player can use the wait command and wipe them out in a single turn.

Forcing robots to collide. Whenever two robots are aligned in a column or row (i.e., having the same x- or y-coordinate), the player has the opportunity to force the robots to collide with one another. In the case of two horizontally aligned robots, the player must be positioned in between the two robots, and either above or below. As long as the player remains above or below the robots, the robots will attempt to converge on the player, and will collide in the process. (The same applies to two vertically aligned robots.) For example, there are two horizontally aligned robots in the lower right-hand corner of Figure 2B. If the player remains at the current location or moves down and to the left, the robots will run into one another before they reach the player. Further, the player can control exactly where the robots will collide by manipulating their speed of convergence. The robots will converge fastest if the player is between the two robots, half as fast if the player is aligned with one of the robots, or not at all if the player is off to one side of the pair.

Lining up robots. Because it is so useful to have robots aligned horizontally or vertically, a good strategy is to try forcing the robots to line themselves up. Even if lining up the robots will not help in the present situation, it may be that after teleporting, the player will be in a position where robots will collide as they converge on him. One simple heuristic for lining up the robots is to march towards the center of mass of the robots. As the robots approach, their movement strategy will tend to place them along the horizontal and vertical axes centered on the player's location.

Teleporting. If the player becomes trapped by robots, teleporting is the only option. However, because there is no guarantee that teleporting will land the player in a "safe" position, the player should avoid teleporting unnecessarily, especially at higher levels of the game.

Collecting a Corpus of Moves

Robots is an addicting game. With practice, it also becomes fairly automatic. One can play while carrying on a conversation or eating. Over a period of several weeks, I played nearly 300 games, and recorded the games to use as examples for RAMBOT. "Recording a game" means saving an image of the board at the start of every turn, along with my move in response to that situation. In the end, 18,200 of these "situation-response" pairs were saved. This corpus did not, of course, represent optimal play; it represented my abilities and included occasional errors, which were not screened out.

RAMBOT

RAMBOT's goal was to learn associations between situations from the corpus and the corresponding responses. That is, given a board image as input, RAMBOT was to produce as output the corresponding move that I made. Following learning, RAMBOT should be capable not only of replicating my performance, but also of generalizing its learning to novel situations: when presented with a situation "similar" to ones it has observed, RAMBOT should suggest a response "similar" to the observed responses.

As in most connectionist networks, input and output representations play a critical role in determining the notion of "similarity," and hence, in determining the sort of generalizations that will be made and the overall difficulty of the learning task. In principle, input and output representations are not important, because with sufficient units in the intermediate layer, any input/output mapping can be achieved. However, practical limitations on the number of intermediate units demand careful selection of input and output representations. With appropriate representations, some of the similarity structure of the game can be built explicitly into the network.

Input Representation

The simplest input representation would be to have two units for each cell on the board. One unit would be turned on if there was a robot in the corresponding cell, the other would be turned on if there was a junk heap, and perhaps both would be turned on if the player was in that cell. However, this scheme has a serious drawback, which can be seen by considering the representations that would be generated for the situations shown in Figures 3A-D. The set of units activated in one situation does not overlap with the set activated in another. Because overlap among input patterns is necessary for the explicit representation of similarity, this encoding does not suggest that the four situations are related, when in fact they are extremely similar: the appropriate response to each situation is to move away from the robots and then stay put, allowing the robots to collide with one another. More generally, a player's response should, for the most part, depend only on the relative location of the robots with respect to the player and each other, not on the absolute location of the player, nor strictly on the absolute distance of the player to the robots, nor on the absolute orientation of the player with respect to the robots (see Figure 3E for an exception). An input representation is required that captures the location,

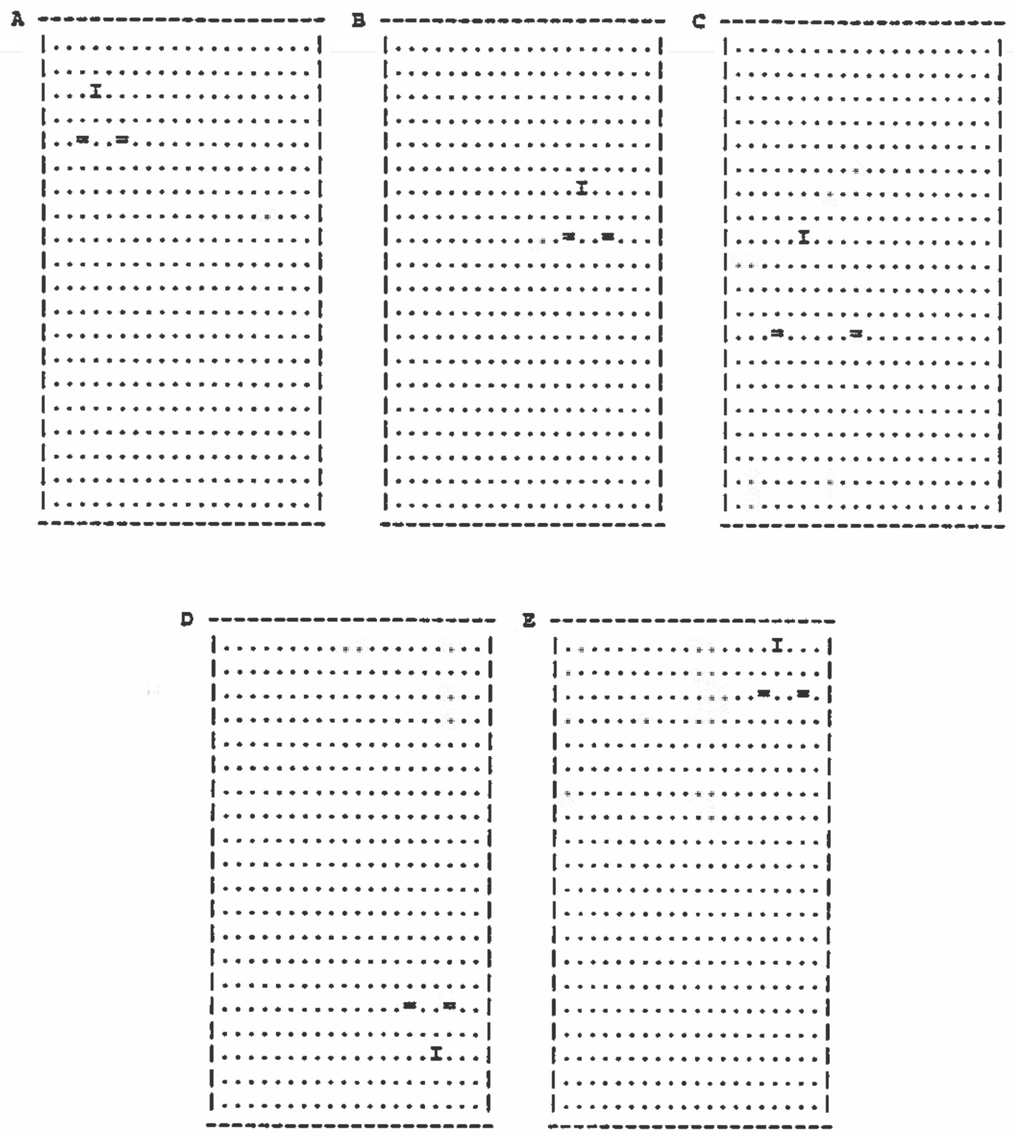


FIGURE 3. Four similar situations (A-D) in which the player's response should be unaffected by absolute location, scale, and orientation of the player with respect to the robots. In one situation (E), the absolute location does matter due to the edge of the board.

scale, and orientation invariances present in the game. The representation chosen for RAMBOT achieves these invariances to a degree. To explain this representation, the steps involved in constructing it from a board image are described.

Force player into upper-left quadrant of board. The board is flipped around so that the player always lies in the upper-left quadrant of the board. If the player is in the upper-right quadrant, the board (and the corresponding move made by the player) is mirrored around the central vertical axis; if the player is in the lower-left quadrant, the board is mirrored around the central horizontal axis; if the player is in the lower-right quadrant, the board is mirrored around both vertical and horizontal axes. Following this procedure, the relative position of the player is fixed with respect to the nearest corner of the board.

Draw windows onto board. Two windows are drawn on the board: a *fine-scale* window, which is centered on the player's location and covers a 13×13 region (6 cells to either side of the player), and a *coarse-scale* window, which is fixed with respect to the player's location and covers a 29×29 region (9 cells to the left of the player, 19 to the right). Because the player is located in the upper-left quadrant, the coarse-scale window is guaranteed to enclose the entire board.

Lay out grid. Within each window, a 7×7 grid of equally spaced points is laid out to span the region within the window. For the fine-scale window, this means that grid points fall on every other cell; for the coarse-scale window, grid points fall on every fourth cell.

Activate units. Each grid point represents the receptive-field center of two input units. One unit is activated by robots in its immediate vicinity, the other by junk heaps. More specifically, the receptive fields of the fine-scale and coarse-scale units are as follows:

0.25	0.50	0.25
0.50	1.00	0.50
0.25	0.50	0.25

0.15	0.19	0.20	0.25	0.20	0.19	0.15
0.19	0.25	0.31	0.50	0.31	0.25	0.19
0.20	0.31	0.45	0.75	0.45	0.31	0.20
0.25	0.50	0.75	1.00	0.75	0.50	0.25
0.20	0.31	0.45	0.75	0.45	0.31	0.20
0.19	0.25	0.31	0.50	0.31	0.25	0.19
0.15	0.19	0.20	0.25	0.20	0.19	0.15

These numbers represent the amount of activation that will be assigned to a unit given that a robot or junk heap appears in various locations with respect to the unit's receptive-field center. For example, if a robot is located at the center of a fine-scale robot-detecting unit's receptive field, 1.0 units of activation will be added to that unit's activation level. If the robot is located in the cell to the immediate lower right of the unit's receptive field center, 0.25 units of activation will be added. Because receptive fields overlap, any object on the board may produce activation in several units. However, the receptive fields are designed to guarantee that the net activity produced by any object is constant, independent of the object's location or the number of receptive fields it lies within.

The walls around the playing field are treated like junk heaps. For practical purposes, they behave the same way—they are inert and the player is not allowed to walk into them. However, because of the large number of points defining each wall, the activity of a wall point was set to only 5% of that produced by a junk heap. The aim was to prevent the presence of walls from overwhelming information about junk heaps.

This representation has many virtues. First, the player can be in various locations on the board, yet the input patterns will look similar if the local arrangement of robots and junk heaps is similar; nonetheless, activations from the walls serve to distinguish cases in which the player is trapped in a corner. Second, because the receptive fields of the units are so broadly tuned, a certain amount of scale invariance is built in. Third, the locations of objects are coarse coded (Hinton, McClelland, & Rumelhart, 1986), meaning that each object activates several nearby units. This helps to define the two-dimensional structure of the board by way of correlations in activity among neighboring units. Fourth, by coding the player's location with respect to the nearest corner, important orientation invariances are captured. Fifth, the two windows onto the board provide both a foveal and global view of the situation, with high resolution in the foveal view.[1]

[1] Two ideas for improving the input representation seem promising but have not been implemented. First, the coarse-scale window wastes a large proportion of its units because they lie off the board. If the window "wrapped around" from one edge of the board to the other, the number of units could be reduced and the remaining units would be better utilized. Second, additional units could be added to the input pattern to represent a temporal context (a time-decaying trace) of previous moves (Jordan, 1985). Thus, the input pattern would specify not only the current board but also a recent history of moves; this would give the network the ability to learn plans extending over time.

Output Representation

The output representation is straightforward. There is one unit for each of the eight "directional" moves, one unit for remaining in the current location, and one unit each for teleport and wait. To provide RAMBOT with explicit information about the arrangement of the directional moves, target output patterns showed activation not only for the selected move, but also for its two "neighbors" (the directional moves to either side of the selected move). For example, when the player responded by moving directly upwards in a given situation, RAMBOT learned to associate that situation with an activity level of 1.0 for the "up" unit, and also with an activity level of 0.2 for the "up-left" and "up-right" units.

Overall Network Structure

The input layer had 196 units, the intermediate layer 74 units, and the output layer 11. The number of intermediate units selected was based on a guess of sufficiency conditions; little work has been done to estimate the necessary number of units. There was full connectivity from one layer to the next, but no direct connections from input to output layers. There were a total of 15,318 connections. The intermediate and output units were semilinear units with a logistic activation function, as described in Rumelhart, Hinton, and Williams (1986).

Evaluation of Performance Using Corpus

RAMBOT has been presented with nearly a million learning trials. This amounts to innumerable hours on our Sun-2's, but only on the order of 12 hours of Cray CPU time. Figure 4 shows performance as a function of learning trial. The bottom line indicates the percent of trials in which the most active output unit corresponded the stored response; the middle line indicates the percent of trials in which either the most active output unit or, if it was a directional move, one of its neighbors corresponded to the stored response; and the top line indicates the percent of trials in which either the most active or second most active output unit corresponded to the stored response.

Performance continues to improve, though the bottom line appears to be approaching an asymptote around 73%. This turns out to be quite impressive, for the following reason. I wrote a program that randomly selected boards from the corpus, displayed them for me, and allowed me to make a new response without knowledge of my original response. Replaying over 10% of the corpus in this fashion, I was able to match my original responses on only 66% of the trials. Thus, RAMBOT is at least as good at predicting my moves as I am.

The graph also shows RAMBOT's ability to generalize. Points labeled with *x*s immediately follow the addition of new moves to the corpus. (The corpus started with only about 4,000 moves and was gradually built up to 18,200.) Performance was barely affected when new moves were added. Thus, RAMBOT is able to respond to unfamiliar moves with almost the same degree of accuracy as to familiar moves. (The drop in performance following a rearrangement of the learning trials, the *r* points, is due to the use of momentum in the back-propagation rule; see Rumelhart et al., 1986.)

In addition to replaying old games, RAMBOT can, of course, play new games. For this purpose, RAMBOT was set up to interact with the robots game. At the start of each turn, an image of the board was encoded on the input units of the network. Activation was allowed to flow through the network to the output units, and the most active output unit was selected as RAMBOT's move. If this move was invalid (i.e., it involved walking into a wall, robot, junk heap, or adjacent to a robot), the move was discarded and the next most active move was considered. The selected move was then fed back to the robots game, the robots were allowed to move, and this cycle repeated.

Although the time required for learning was substantial, play proceeds in real time. Move selection takes about 1-2 seconds on a Sun-2 with a floating point board. Watching RAMBOT play is impressive. Most of the time it does just what I would have done—a clever program indeed.

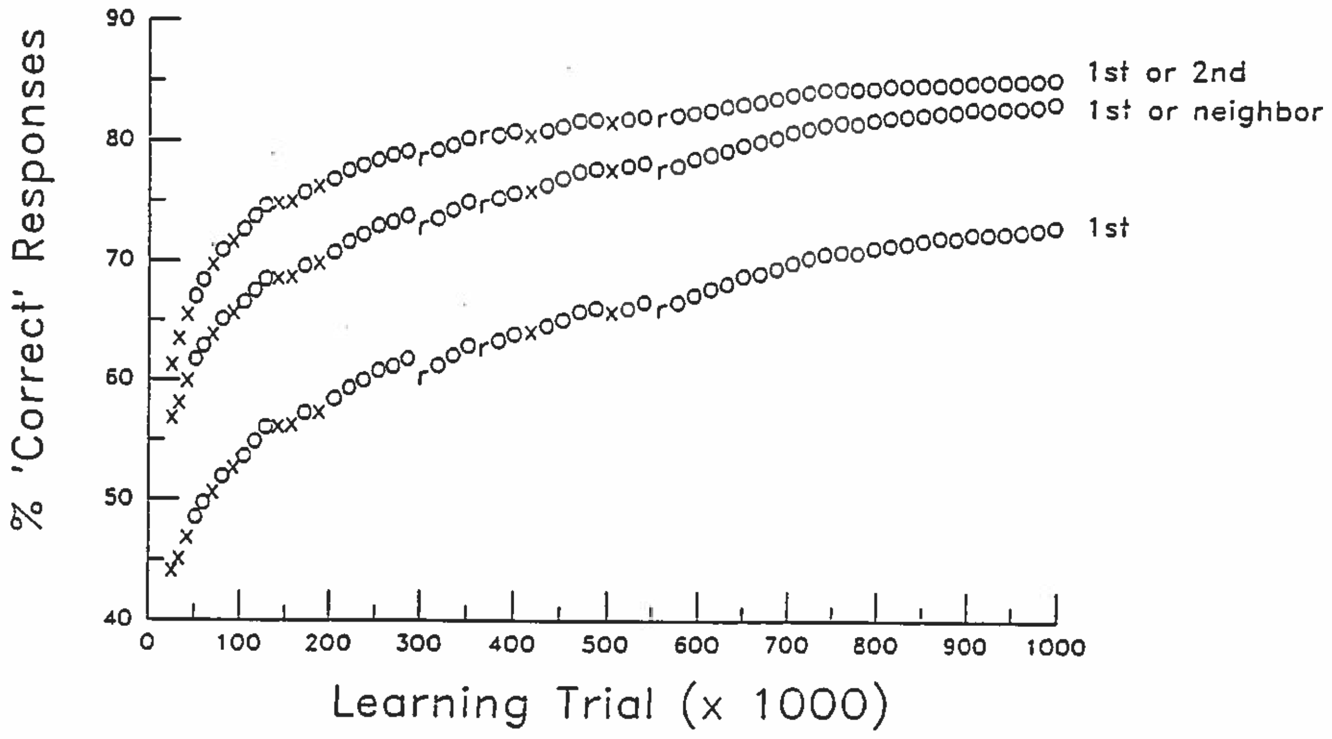


FIGURE 4. Performance as a function of learning trial. The bottom line indicates the percent of trials in which the most active output unit corresponded to the stored response; the middle line indicates the percent of trials in which either the most active output unit or, if it was a directional move, one of its neighbors corresponded to the stored response; and the top line indicates the percent of trials in which either the most active or second most active output unit corresponded to the stored response. *x* = new moves added to corpus; *r* = reordered presentation sequence.

Examples From Play

What follows are several typical examples from actual play.

Getting ready to teleport. Figure 5 shows the board at the start of a turn, as well as the activation levels of the output units in response to that board. The 3×3 array of numbers indicates the activity levels of the eight directional moves and the remain-in-current-location move, arranged by direction. The letter *t* stands for the teleport unit, *w* for the wait unit. Activation levels range from 0-1. The activation level of the selected move is flagged by an asterisk. In Figure 5, RAMBOT is trapped and its only valid move is to teleport. This is the move with the highest activation level. It is interesting to note that three other moves receive some activation: down, down-left, and left. These are the moves that one would consider if the wall were not present. Thus, this example shows that RAMBOT has learned certain facts about the game: walking into walls and robots is not an option, and when being chased by robots, move away from them.

Forcing robots to collide. Figure 6 shows a sequence of moves in which RAMBOT comes around from the right of two horizontally aligned robots and forces them to collide with one another.

Hiding behind a junk heap. Figure 7 shows a sequence of moves in which RAMBOT uses a junk heap to protect itself. RAMBOT first moves towards the junk heap, forcing the robot to its right to crash into the heap, then moves above the heap, forcing the robots below to crash into the heap.

```
 ----------------------
|....................|
|....................|
|....................|
|....................|
|....................|
|....................|
|....................|          0.00    0.00    0.00
|....................|
|....................|          0.21    0.00    0.00
|....................|
|....................|          0.38    0.23    0.00
|=...................|
|......=.............|
|I==......@=.........|          t:  *0.59
|....................|
|........@...........|          w:   0.00
|....................|
|....................|
|....................|
|....................|
 ----------------------
```

FIGURE 5. The current board and RAMBOT's choice of a move. The numeric values on the right are activation levels of the output units and are explained in the text.

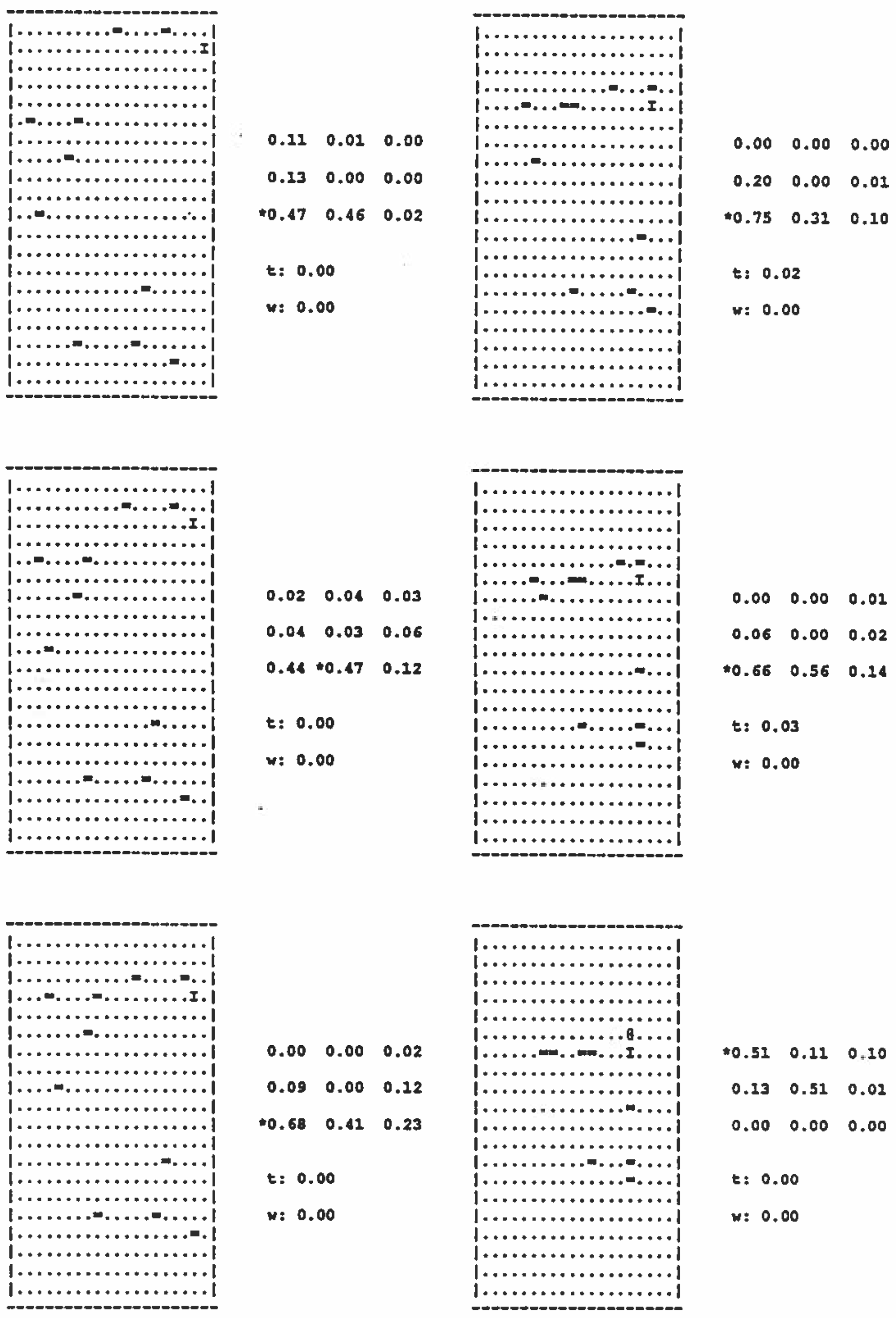
Shortcomings. In observing RAMBOT at play, only two strategic shortcomings were evident to me. First, RAMBOT tended to teleport when being chased by robots and a wall was approaching as shown in Figure 8A. Generally, this is a reasonable strategy, but in the example shown, it appears that RAMBOT teleported too soon. It could have forced the two aggressor robots to collide instead, thereby altering the game state radically. Unfortunately, the tendency to teleport prematurely has serious consequences: one teleport often leads to another, and each teleport is life threatening.

Second, RAMBOT appears to have overgeneralized the conditions under which the wait command applies. Waiting is a good move when the player is protected by a junk heap and all robots are lined up on the other side of the heap. However, as shown in Figure 8B, RAMBOT did not pay serious enough attention to the presence of robots elsewhere on the board. I should confess that "waiting in the face of danger" is an error I occasionally make, and several instances of this behavior appear in the learning corpus.

Evaluation of Performance in Free Play

In order to evaluate RAMBOT's performance in free play, two statistics were collected over a thousand game sample. These statistics were (a) the level at which the player died, and (b) the average number of moves required to successfully complete a level. For comparison, statistics were also collected on my play in the 300 game learning corpus and a random strategy over a thousand game sample. The random strategy, RANDOM, chose its move entirely at random from the set of valid moves.

Level of death. When all robots on the board are destroyed, the game restarts at a higher level of difficulty, with an increasing density of robots at higher levels. On average, RANDOM reached level 2.10, RAMBOT level 2.97, and I managed to get to level 4.33. To determine the effect of RAMBOT's erroneous "waiting in the face of danger" (see Figure 8B), the wait command was replaced by the remain-still-for-one-move command and another thousand games were run with this modified strategy. On average, RAMBOT without the wait command reached level 3.57, significantly better than with wait. Clearly, overgeneralization of the conditions under which this command applies had a dramatic effect on performance.



```
 0.11  0.01  0.00

 0.13  0.00  0.00

*0.47  0.46  0.02

t: 0.00

w: 0.00
```

```
 0.00  0.00  0.00

 0.20  0.00  0.01

*0.75  0.31  0.10

t: 0.02

w: 0.00
```

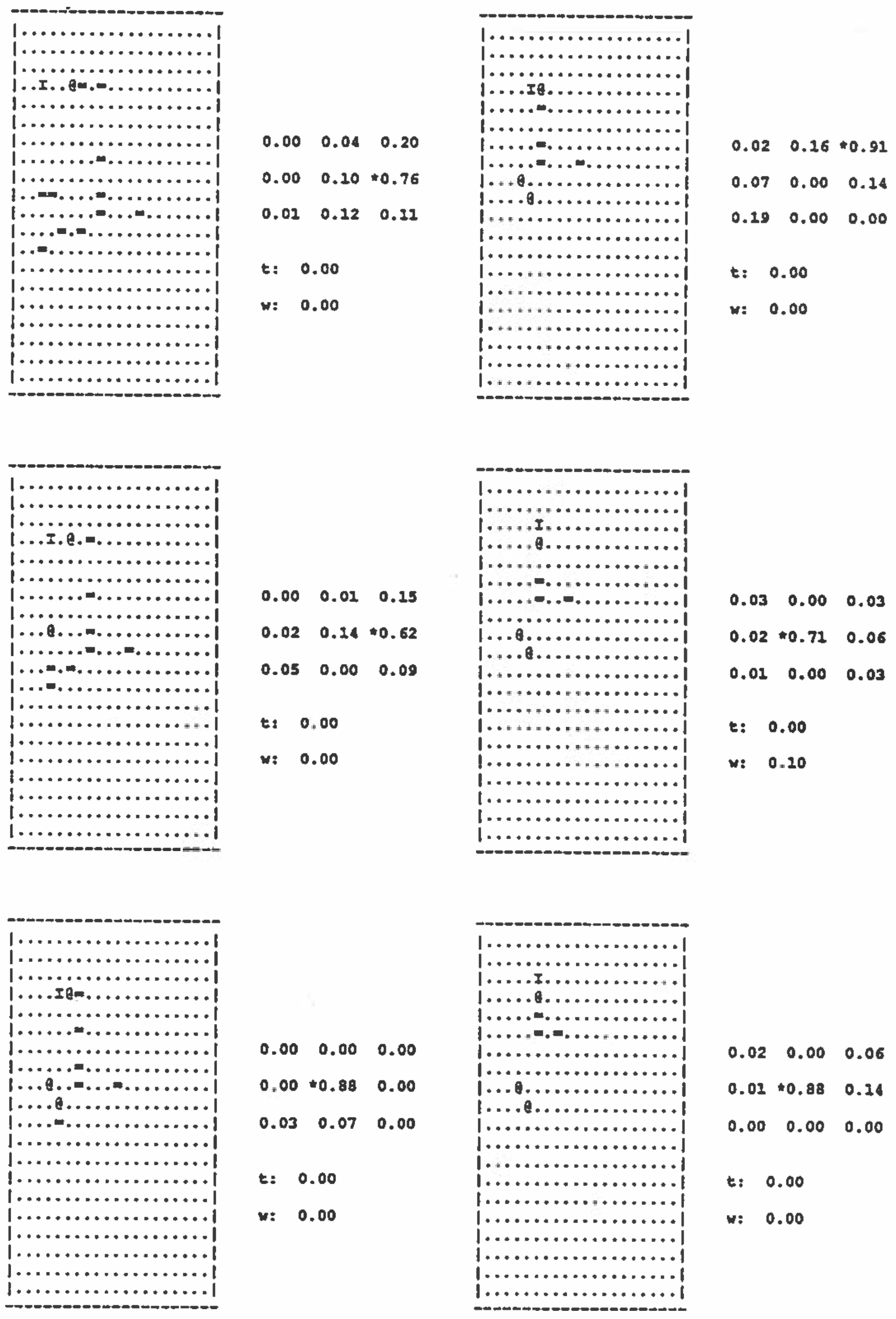
```
0.02  0.04  0.03

0.04  0.03  0.06

0.44 *0.47  0.12

t: 0.00

w: 0.00
```

```
 0.00  0.00  0.01

 0.06  0.00  0.02

*0.66  0.56  0.14

t: 0.03

w: 0.00
```

```
 0.00  0.00  0.02

 0.09  0.00  0.12

*0.68  0.41  0.23

t: 0.00

w: 0.00
```

```
*0.51  0.11  0.10

 0.13  0.51  0.01

 0.00  0.00  0.00

t: 0.00

w: 0.00
```

FIGURE 6. A sequence of six moves in which RAMBOT forces two robots to collide. The sequence begins in the upper-left corner, moves down the column, and then on to the right-hand column.



```
0.00  0.04  0.20
0.00  0.10 *0.76
0.01  0.12  0.11

t:  0.00
w:  0.00
```

```
0.02  0.16 *0.91
0.07  0.00  0.14
0.19  0.00  0.00

t:  0.00
w:  0.00
```

```
0.00  0.01  0.15
0.02  0.14 *0.62
0.05  0.00  0.09

t:  0.00
w:  0.00
```

```
0.03  0.00  0.03
0.02 *0.71  0.06
0.01  0.00  0.03

t:  0.00
w:  0.10
```

```
0.00  0.00  0.00
0.00 *0.88  0.00
0.03  0.07  0.00

t:  0.00
w:  0.00
```

```
0.02  0.00  0.06
0.01 *0.88  0.14
0.00  0.00  0.00

t:  0.00
w:  0.00
```

FIGURE 7. A sequence of six moves in which RAMBOT uses a junk heap to protect itself.

```
A ----------------------
  |....................|
  |....................|
  |....................|
  |....................|
  |....................|
  |......=.............|
  |............=.......|   0.00    0.00    0.20
  |....................|
  |.......=.......=I...|   0.00    0.00    0.26
  |...............=....|
  |....................|   0.00    0.00    0.35
  |....................|
  |...........=........|
  |..........=.........|   t:  *0.47
  |........=...........|
  |....................|   w:   0.00
  |..........=.....=...|
  |....................|
  |....................|
  |....................|
  ----------------------
```

```
B ----------------------
  |....................|
  |....................|
  |....................|
  |....................|
  |....................|
  |.............@......|
  |....................|   0.34    0.15    0.09
  |....................|
  |......=.............|   0.02    0.22    0.01
  |........==.==.@I....|
  |....................|   0.00    0.00    0.00
  |....................|
  |...............=....|
  |....................|   t:   0.00
  |....................|
  |....................|   w:  *0.85
  |....................|
  |....................|
  |....................|
  |....................|
  ----------------------
```

FIGURE 8. Examples of poor moves chosen by RAMBOT.

Average moves per completed level. Table 1 shows, by level, the average number of moves per successfully completed level. RANDOM never got past level 5 and required over twice as many moves as either RAMBOT or myself, but RAMBOT's performance was comparable to mine. Because successful completion of a level is often prevented by the two shortcomings noted earlier (see Figure 8), Table 1 provides some indication of RAMBOT's playing abilities when such basic errors are not made.

BEYOND RAMBOT

RAMBOT appears to have captured at least some of my expertise in the game of robots. RAMBOT could no doubt be improved, perhaps by adding moves to the learning corpus, by modifying the input representation (see Footnote 1), by increasing the number of intermediate units, or by adding direct input/output connections. Further possibilities include constructing a connectionist network that learns by *experience* using reinforcement learning techniques (Barto & Anandan, 1985; Barto, Sutton, & Brouwer, 1981); or constructing an optimal robots-playing program which uses brute force search techniques and teaching RAMBOT with examples from this optimal program. While both of these approaches are feasible for the relatively simple game of robots, they are far less so for complex domains such as chess playing and medical diagnosis. The goal of RAMBOT was not to build an optimal robots-playing system, but to build a system that started with little knowledge about a domain and could learn general rules of the domain by observing a human expert.

TABLE 1

Level	RANDOM	RAMBOT	MIKE
1	57.1	21.9	18.0
2	48.9	21.2	19.6
3	44.6	21.1	18.7
4	43.3	20.6	20.8
5	47.0	21.7	22.8
6	—	19.8	22.8
7	—	20.0	22.0

What are the costs and benefits of building such a system with connectionist techniques? To begin with, domain experts must specify every source of information that is potentially relevant to their decision processes (though they need not specify how the information is used). This information serves as the input to the connectionist net. Further, experts must provide a corpus of performance data, sufficiently large to sample the input space well. Without representative sampling, the system will not have solid ground on which to base generalizations.

The one serious drawback to a connectionist expert system is that the system itself has little power of explanation. It is possible to examine the outputs of the intermediate units, and to use the "internal representations" developed by the system as a justification for decisions, but generally these internal representations are so complex and highly distributed that they simply add to confusion rather than help to explain the system's behavior. A more reasonable means of increasing the explanatory power of the system is to break down its decision process. For example, in the case of medical diagnosis, the appropriate input/output mapping would not be from symptoms to diseases, but from known symptoms to possible diseases *and* further tests that could be performed to discover additional symptoms. Thus, the system could be used iteratively, performing tests suggested by the network and then feeding results of these tests back into the system. This approach at least provides a sequence of steps taken by the system to reach a decision.

RAMBOT does illustrate several important and unique properties. First and foremost, the system is able to generalize from training examples. Second, the system is able to learn behavior that is dependent on an extremely large number of variables—the robots playing board contains 400 cells—and is able to learn despite inconsistent expert behavior, as my inability to reproduce moves in the learning corpus attests to. Third, the system is able to suggest multiple hypotheses with varying degrees of certainty, as embodied by the activation levels of the output units. Fourth, the system allows for the nonlinear combination of evidence, unlike many expert systems that use numerical methods (Charniak & McDermott, 1985).

Beyond these generalities, what does the success of RAMBOT have to suggest for the construction of learning connectionist expert systems in other domains? One problem with the robots game is that it can easily be thought of as a perceptual, pattern-matching task. Connectionist systems are commonly held to be good at this sort of task, but it is not as clear that connectionist techniques will prove useful in "higher-level," symbolic domains. As an argument against this point of view, consider a domain far removed from perception: using a computer operating system, say UNIX. A connectionist expert system for this domain is feasible. The idea would be to build a UNIX apprentice program (UNIXBOT?) that could learn to predict what command the user was likely to type next based on a recent history of commands and some contextual information, such as the time of day or the current working directory. If the system could make strong enough predictions, it could correct user errors, or even anticipate commands.

In principle, a system that learns to predict what command will be typed next is no different than one that learns to predict the next move of a game. It seems that much of cognitive behavior can be framed in terms of pattern recognition, even though we don't ordinarily think of that behavior as being perceptual. Experts in a domain just "see" solutions (Rumelhart, 1984). If this is indeed true, connectionist techniques may have application to a wide range of expert systems applications.

REFERENCES

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

Anderson, J. A., & Hinton, G. E. (1981). Models of information processing in the brain. In G. E. Hinton & J. A. Anderson (Eds.), *Parallel models of associative memory.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Barto, A. G., & Anandan, P. (1985). Pattern recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics, 15,* 360-375.

Barto, A. G., Sutton, R. S., & Brouwer, P. S. (1981). Associative search network: A reinforcement learning associative memory. *Biological Cybernetics, 40,* 201-211.

Charniak, E., & McDermott, D. (1985). *Introduction to artificial intelligence.* Reading, MA: Addison-Wesley.

Duda, R. O., & Shortliffe, E. H. (1983). Expert systems research. *Science, 220,* 261-268.

Hinton, G. E., McClelland, J. L., & Rumelhart, D. E. (1986). Distributed representations. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group, *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press/Bradford Books.

Jordan, M. I. (1985). *The learning of representations for sequential performance.* Unpublished doctoral dissertation, University of California, San Diego.

Rumelhart, D. E. (1984, October). *The nature of expertise.* General discussant at the meeting on The Nature of Expertise, Pittsburgh, PA.

Rumelhart, D. E., Hinton, G. E., & Williams, R. (1986). Learning internal representations by error propagation. In D. E. Rumelhart, J. L. McClelland, & the PDP Research Group, *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press/Bradford Books.

Rumelhart, D. E., & McClelland, J. L. (1986). On learning the past tenses of English verbs. In J. L. McClelland, D. E. Rumelhart, & the PDP Research Group, *Parallel distributed processing: Explorations in the microstructure of cognition. Vol. 2: Psychological and biological models.* Cambridge, MA: MIT Press/Bradford Books.