A Parallel Network that Learns to Play Backgammon

G. Tesauro*
Center for Complex Systems Research, University of Illinois at Urbana-Champaign, 508 S. Sixth St., Champaign, IL 61820, U.S.A.

T.J. Sejnowski**
Biophysics Department, The Johns Hopkins University, Baltimore, MD 21218, U.S.A.

ABSTRACT

A class of connectionist networks is described that has learned to play backgammon at an intermediate-to-advanced level. The networks were trained by back-propagation learning on a large set of sample positions evaluated by a human expert. In actual match play against humans and conventional computer programs, the networks have demonstrated substantial ability to generalize on the basis of expert knowledge of the game. This is possibly the most complex domain yet studied with connectionist learning. New techniques were needed to overcome problems due to the scale and complexity of the task. These include techniques for intelligent design of training set examples and efficient coding schemes, and procedures for escaping from local minima. We suggest how these techniques might be used in applications of network learning to general large-scale, difficult "real-world" problem domains.

1. Introduction

There has been a tremendous resurgence of interest in computing with massively parallel fine-grain architectures, particularly with "connectionist" or "neural" networks. This is to a large extent due to several recent learning algorithms [2, 17, 22, 32, 38, 39] which have overcome some of the problems of earlier architectures [29, 36, 46]. These learning algorithms have demonstrated considerable success at general-purpose learning of small-scale computational tasks, and there are several reasons to believe that they might prove useful for

* To whom reprint requests should be addressed. Present address: IBM Watson Labs., P.O. Box 704, Yorktown Heights, NY 10598, U.S.A.
** Present address: The Salk Institute, P.O. Box 85800, San Diego, CA 92138, U.S.A.

certain classes of larger-scale problems that are encountered in the real world. Problem domains which are currently being studied include English pronunciation [41], speech recognition [7, 33], protein folding [31], medical diagnosis [23], sonar target identification [13], and 3-D shape analysis of shaded images [24].

Connectionist learning procedures show particular promise in domains where there are many graded features that collectively contribute to the solution of a problem. In the connectionist approach, learning proceeds by making small numerical adjustments to a large number of parameters rather than a few discrete changes to the relationships between Boolean categories. This allows efficient techniques from numerical analysis to be employed in optimizing performance. In contrast, methods based on symbolic inductive inference are more appropriate for domains where categories are well defined and knowledge is highly structured [28].

A useful testing ground for studying issues of knowledge representation and learning in networks can be found in the domain of game playing. Board games such as chess, go, backgammon, and Othello entail considerable sophistication and complexity at the advanced level. Mastery of expert concepts and strategies often takes years of intense study and practice. Furthermore, a myriad variety of skills is often called upon, including extensive memorization of opening variations, intricate calculation of tactical exchanges, intuitively sensing the long-range potential of a current board configuration, and even assessing the psychological state of one's opponent. The complexities in board games, however, are embedded in relatively "clean" structured tasks with well-defined rules of play, and well-defined criteria for success and failure. This makes them amenable to automated play, and in fact most of these games (particularly chess) have been extensively studied with conventional computer science techniques [11, 25]. Thus, direct comparisons of the results of network learning can be made with more conventional approaches.

The choice of a particular game of study should depend on the particular types of skills required for successful play. We make a distinction between two fundamentally different kinds of skills. First, there is the ability to "look ahead," i.e., to work out the future consequences of a current state, either by exhaustive tree search, or by more symbolic, deductive reasoning. Secondly, there is "judgmental" ability to accurately estimate the value of a current board state based on the patterns or features present, without explicitly calculating the future outcome. The game of backgammon is unusual amongst games because the judgmental aspects predominate, in comparison with other games like chess that often require look ahead to great depth. The principal reason for this is the probabilistic element in backgammon: each move depends on a roll of the dice. There are 21 distinct dice rolls and around 20 possible legal moves for each roll. Tree search algorithms would thus be inappropriate, since the average branching factor at each ply of the search would be about 400. As a consequence most backgammon programs and most humans do not search more than one or two ply, but rather rely on pattern recognition and judgmental skills.

In simple terms, backgammon is a one-dimensional race to the finish, involving both skill and luck. Play proceeds by rolling the dice and moving pieces forward according to the dice roll. The full complexity of the game emerges in "engaged" positions, i.e., positions in which it is necessary to cross over some of the opponent's pieces in order to reach the finish. In such positions it is possible to "hit" an opponent's piece and send it back to the starting position, and to form a "blockade" which impedes the forward progress of the opponent's pieces. These possibilities lead the expert to develop a number of advanced concepts and strategies of considerable complexity. In contrast, in "racing," or disengaged positions, hitting and blocking are not possible, and the computation of correct moves is much simpler. Additional complications are introduced through a "doubling cube" which a player can use to force his opponent to either resign or accept a doubling of the stakes of the game. A glossary of backgammon terminology is provided in an appendix. For a more detailed description of the rules, strategies, etc., we refer the reader to [26].

Of particular interest to the present study is Hans Berliner's backgammon program BKG [1, 4–6], which has reached an advanced level of performance. BKG uses an evaluation function containing a large number of hand-crafted features which were devised based on the knowledge of human experts. The features measure quantities such as mobility and the probability of being hit. The contribution of each feature to the total evaluation is weighted by an "application coefficient" which measures the degree of relevance of the feature in the particular situation. These application coefficients vary smoothly as the position is changed, and depend themselves on the hand-crafted features, so the overall evaluation function is a nonlinear function of the features. BKG provides an existence proof that a static evaluation function capturing much of human expert knowledge can be produced. However, since machine learning techniques did not contibute to the evolution of BKG, the question of whether such a sophisticated evaluation function could be *learned* was not addressed.

One of the first demonstrations of machine learning in a games environment was Samuel's checkers program [40]. This program's evaluation function was simply a linear combination of a set of hand-crafted features, with constant coefficients adjusted in learning. Learning was based on a comparison of the evaluation function for a particular position with the value of the expected future position a certain number of moves ahead obtained by following the most plausible line of play. If the expected future value was greater than the current value, then the coefficients of the features contributing positively to the evaluation function were increased, and the coefficients of the negatively contributing features were decreased. Conversely, if the expected future value was less than the current value, the coefficients were changed in the opposite fashion. In addition to gradual changes in the coefficients, abrupt changes could also occur when one feature was replaced with another. One fundamental limitation of this approach is that all of the features used in the evaluation function came from a list drawn up by the human programmer; the more difficult problem of discovering and defining new features was not addressed in this study.

Relatively little work has been done to explore learning in games since Samuel's pioneering efforts in checkers. A notable exception is the recent work of Frey et al. on Othello [12, 30], which uses modern computational resources to study learning in very much the same spirit as in Samuel's approach. Once again, a linear evaluation function was constructed using a set of heuristic hand-crafted features. A large set of late middle game positions taken from tournament play was used to fit the heuristic evaluation function to the exact values obtained by exhaustively searching the tree of possible moves from each position to the end of the game. While this is an effective procedure for developing an accurate evaluation function, Frey expresses disappointment that, more than two and a half decades after Samuel's original work, there is still no known procedure for automating the definition of features.

We have used a connectionist approach to study learning of a sophisticated backgammon evaluation function. Specifically, we have used a deterministic, feed-forward network with an input layer, an output layer, and either one or two layers of hidden units. The algorithm used to train the network is the so-called "back-propagation" algorithm [22, 32, 38, 39]. Our choice of backgammon was motivated by the considerations of judgment versus look-ahead discussed previously. Connectionist networks are believed to be much better at judgmental tasks than at tasks involving sequential reasoning, and thus would appear to be well-suited to the domain of backgammon. Our learning procedure is a supervised one that requires a database of positions and moves that have been evaluated by an expert "teacher." In contrast, in an unsupervised procedure [18, 40, 42], learning would be based on the consequences of a given move (e.g., whether it led to a won or lost position, as in [3]), and explicit teacher instructions would not be required. However, unsupervised learning procedures thus far have been much less efficient at reaching high levels of performance than supervised learning procedures. In part, this advantage of performance than supervised learning can be traced to the higher quantity and quality of information available from the teacher. (Of course, when learning is based solely on teacher instructions, then the student cannot surpass the teacher, except perhaps in thoroughness and endurance. The best one could realistically hope for is a network which plays as well as the teacher that trained it.)

Some of the issues that are explored in this study are important general issues in connectionist learning, and have also been discussed by other authors [16, 23, 27, 43]. Amongst the most important are scaling and generalization.

Most of the problems that have been examined with connectionist learning algorithms are relatively small-scale and it is not known how well they will perform on much larger problems. Generalization is a key issue in learning to play backgammon since it is estimated that there are 10^{20} possible board positions, which is far in excess of the number of examples that can be provided during training. In terms of both the ratio of training set size to total set size, and the predicate order of the computational task (more will be said about this later), we believe that our study is the most severe test of generalization in any connectionist network to date.

We have also identified in this study a novel set of special techniques for training the network which were necessary to achieve good performance. A training set based on naturally occurring or random examples was not sufficient to bring the network to an advanced level of performance. Intelligent database design was necessary. Performance also improved when noise was added to the training procedure under some circumstances. Perhaps the most important factor in the success of the network was the method of encoding the input information. The best performance was achieved when the raw input information was encoded in a conceptually significant way, and a certain number of pre-computed features were added to the raw information. These lessons may also be useful when connectionist learning algorithms are applied to other difficult large-scale problems.

2. Operational Paradigm

The first step in designing any connectionist system is to decide what input-output function it is to compute. In general, it is not necessary for a single network to handle all aspects of game play. A complete game-playing system might use several networks, each specialized for a particular aspect of the problem. In the present study, we have focused solely on move-making decisions, and have not considered the problem of making doubling decisions. Furthermore, we have only trained the network on engaged positions, since the decisions for racing positions are much easier and can be handled very effectively by a simple algorithm.

For many problems, such as text-to-speech conversion, the network inputs and outputs can be identical to the inputs and outputs in the formal statement of the computational task. In backgammon the formal input is a description of a board position and roll, and the desired output is a move (or equivalently a final board position). We would expect it to be extremely difficult to teach a network this input-output function, primarily because an enormous amount of effort would have to be expended in teaching the network the constraint of move legality. (This is due to two factors: the computation of legal moves, if expressed as a Boolean predicate, would be of very high order, and the constraint of legality is severe, i.e. the output must be exactly legal, whereas connectionist networks tend to find only approximate solutions for hard problems.) Additional problems would be expected in situations in which there is no clear best move. Competing moves of approximately equal value could cause serious interference.

These problems can be surmounted by teaching the network to *select* moves, rather than *generate* them. We envision our network operating in tandem with a pre-processor which would take the board position and roll as input, and produce all legal moves as output. The network would be trained to score each move, and the system would choose the move with the highest network score.

There are two possible ways of training the network to score moves. One would be to present each final board position to the network, and train it to produce a numerical output corresponding to the absolute value of the position (e.g., the expected payoff at the end of the game). The problem with this approach is that the network is to be trained on the judgment of human experts, and it is very difficult for human beings to assign absolute numbers for the overall value of a given position. The other approach would be to present both the initial and final positions to the network, and train it to produce a relative "strength of move" output. This approach would have greater sensitivity in distinguishing between close alternatives, and corresponds more closely to the way humans actually evaluate moves. For these reasons we have chosen the latter operational paradigm. These competing alternatives are summarized in Table 1.

3. Construction of the Training Set

We now discuss the construction of a training set conforming to the operational paradigm discussed previously. Each line in the database contains a board position, a roll, a suggested move, and a human expert's judgment of the strength of that move. The expert judgment is expressed as an integer in the range [−100, +100], with +100 representing the best possible move and −100 representing the worst possible move.

Table 1
A summary of three different types of operational paradigms for backgammon. In each the input includes dice roll information. (a) The input is the initial board state, and the desired output is a move, or equivalently a final board state. (b) The input is a final board state, and the desired output is a score indicating the absolute value of that state. (c) The input contains both the initial and final board states, and the output is a score indicating the relative strength of the move.

Input	Output
(a) Initial position	Move
(b) Final position	Absolute score
(c) Initial and final position	Relative score

Example 3.1. Consider the following sample entry in the 3200-position training set.

```
1-2, 6-(−5), 8-(−3), 12-5, 13-(−5), 17-3, 19-5, 24-(−2); W65
/1−7,7−12 = 100
/1−7,12−17 = 0
/1−7,17−22 = 0
/12−18,12−17 = 60
/12−18,17−22 = 0
/12−18,18−23 = 0
/17−22,17−23 = −100
```

The first line contains integers describing the initial board position in the format *x*-*n*, where *x* is the board location and *n* is the number of men at the location (with positive *n* indicating White men and negative *n* indicating Black men). Each of the 7 subsequent lines lists a legal move of the particular dice roll (White to play 6–5), followed by the score assigned by the human expert. The human expert has said that the move 1–7,7–12 is the best possible move, 12–18,12–17 is a moderately strong alternative, and 17–23,17–22 is a horrible blunder. The other legal moves have not been judged by the expert, so they are assigned a score of zero.

In general, it is not feasible for a human expert to comment on all possible moves, so our approach is to record the expert's comments on only the few moves he considers relevant, and to leave the remaining moves unscored. (Our database thus consists primarily of unscored lines of data. The handling of these unscored lines in the training procedure will be discussed in Section 5.) In addition, it is important to have examples of bad moves in the database, which are not usually mentioned in expert commentary. The bad moves are necessary because otherwise the network would tend to score all moves as good moves. We have tried to arrange our database so that for each group of moves generated from a given position, there is at least one example of a bad move.

Our current database contains a total of 3202 board positions. All of these positions involved some degree of engagement or contact between the forces, and only a handful of positions involved bear-off situations. One of us (G.T.) is a strong backgammon player, and played the role of human expert in entering scores for the moves in each of these positions. Several different sources were used for the positions in our training set. Approximately 1250 positions were taken from various backgammon textbooks [8–10, 19, 20, 26]. The scores for these moves tend to reflect the opinions of the respective authors, although some effort was made to impose overall consistency on the resulting data. About 1000 positions came from games in which G.T. played both sides. 500 positions were from games in which G.T. played against the network, and 300 positions came from games in which the network played against itself. In the latter two cases, the positions selected were positions in which the network was judged to have handled the position incorrectly. Finally, a total of about 150 hand-crafted positions were included in an attempt to correct specific categories of mistakes that the network had made previously.

As an example of the use of intelligently designed positions to correct specific problems, we illustrate in Fig. 1 a position in which White is to play 6–5. This is a late holding game position in which White is substantially behind. The correct move is 4–10,12–17, hitting the Black blot and sending it back behind White's blockade. The network does in fact give this move a score of 0.95 on a scale from 0 to 1. The problem is that the network prefers the alternative move 16–22,17–22, a move which does not hit, but which makes an additional point in White's inner board. This move is scored 0.98 by the network. This problem demonstrates in a nutshell the basic strengths and weaknesses of our supervised learning approach. The network appears to have learned to judge moves based on a weighted summation of visually appealing features. Such analysis can be of considerable sophistication, but the network suffers from an inability to reason logically about the consequences of its

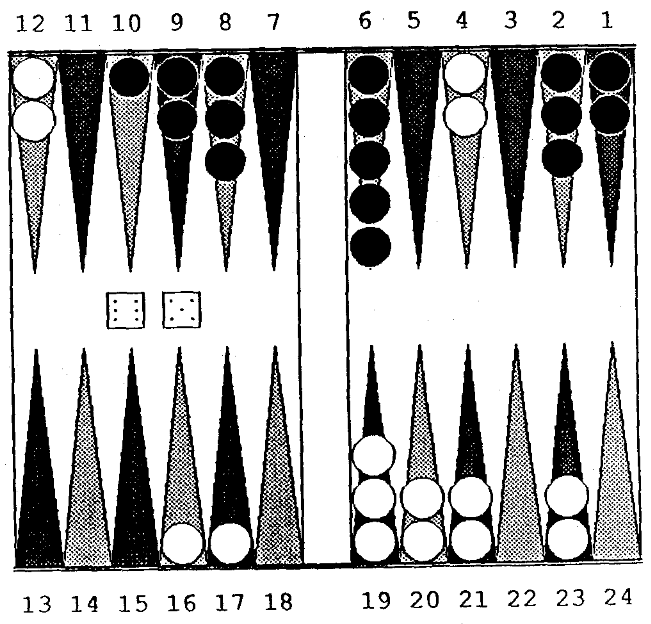

Fig. 1. A sample position taken from [26] illustrating a characteristic defect of the network which is remedied by hand-crafted examples in the training set. White is to play 6–5. The correct move is 4–10,12–17. The network prefers 16–22,17–22. The graphic display was generated on a Sun Microsystems workstation using the Gammontool program.

actions, and to simultaneously consider two competing moves. Humans have little trouble with this position because we can foresee that White is not likely to get another chance to hit, and can deduce that without hitting, the game will evolve after several moves into a pure racing situation in which White has virtually no chance to win. Therefore a hit is imperative, even if the resulting position is somewhat awkward or vulnerable. The network has no such reasoning capability, and thus has trouble finding the right move. Humans are also able to consider the hitting and nonhitting moves together, and to reject the nonhitting move because of the availability of the hitting alternative. The network, however, can only consider one move at a time, and thus does not realize that 16–22,17–22, which would ordinarily be the correct move if no hit were possible, is actually wrong here because of the hitting alternative.

It is also instructive to examine the steps needed to correct this particular deficiency of the network. If one simply includes a large negative score for 16–22,17–22 in the training set, one finds that the network refuses to learn this judgment. There are so many other examples in the training set in which this type of move has been scored extremely high that the network does not learn to produce a low score in this particular instance. Instead, it is necessary to add a set of about 20 hand-crafted examples of positions of this characteristic type (i.e. late game holding positions in which there is a choice between hitting and making a very good point in one's inner board) to the training data. After this number of examples, the network finally catches on to the general idea that hitting takes precedence over filling the inner board in late holding game situations.

In summary, we emphasize the necessity of intelligently hand-crafting example positions to include in the training set which illustrate particular points. It would not be possible, we claim, for the network to obtain this information simply from a training set consisting of a large number of randomly accumulated, undesigned positions. This is because the particular situation in question occurs extremely infrequently during the course of regular game play. In most holding game positions, there is no hit available, so the correct move is to fill in one's inner board while waiting. A small fraction of the time, a hit will be possible, but the alternative filling move will not be particularly attractive. In only an extremely small fraction of positions will there be a *choice* between a hitting move and a very good filling move. Thus the network is not likely to learn this particular principle from random positions alone.

4. Network Design

As stated previously, we use either a three-layer or four-layer network of deterministic analog units. The networks are fully connected between adjacent layers of units. (We have tried a number of experiments with restricted receptive fields, and generally have not found them to be useful.) Since the desired output of the network is a single real value, only one output unit is required.

The coding of the input patterns is probably the most difficult and most important design issue. We have experimented with a large number of coding schemes, and expect our representation scheme to continue to evolve as we gain more experience with the network. In its current configuration the input layer contains 459 input units. A location-based representation scheme is used, in which a certain number of input units are assigned to each of the 26 locations[1] on the board. The information in the database is inverted if necessary so that the network always sees a problem in which White is to play.

An example of the coding scheme used until very recently is shown in Fig. 2. This is essentially a unary encoding of the number of men at each board location, with the following exceptions: First of all, the units representing 5 men at a given location can take on multiple values to encode those rare cases when there are more than 5 men present. (Specifically, if x is the number of men and $x \geq 5$, we set the value of the unit to $x - 4$.) Secondly, the unit representing two men is set to 1 whenever two or more men are present. (This is because all of these cases share the common property of being a "point" that the opponent cannot land on.) Finally, the units representing the final position are turned on only if there has been a change in the number of men at the given location. This was implemented to give the network added sensitivity in

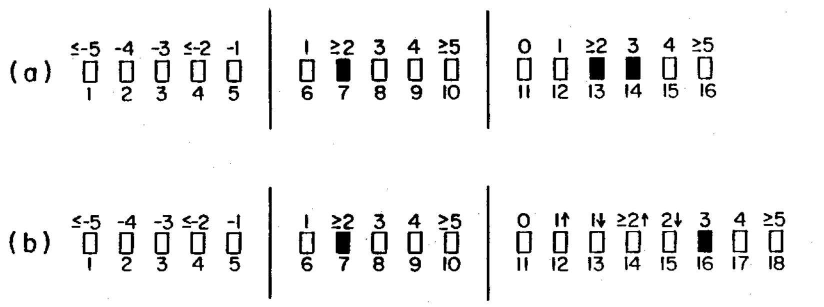

Fig. 2. Two schemes used to encode the raw position information in the network's input. Illustrated in both cases is the encoding of two White men present before the move, and three White men present after the move. (a) An essentially unary coding of the number of men at a particular board location. Units 1–10 encode the initial position, units 11–16 encode the final position if there has been a change from the initial position. Units are turned on in the cases indicated on top of each unit, e.g., unit 1 is turned on if 5 or more Black men are present initially, etc. (b) A superior coding scheme with more units used to characterize the type of transition from initial to final position. An up arrow indicates an increase in the number of men, a down arrow indicates a decrease. Units 11–15 have conceptual interpretations: 11 = "clearing," 12 = "slotting," 13 = "breaking," 14 = "making," 15 = "stripping" a point.

[1]There are 24 basic board locations, plus White bar and Black bar. We have not included an explicit representation of the number of men taken off the board.

distinguishing between competing moves, and has the further advantage of reducing the number of input units required. If the final position is different from the initial position, it must be either zero or positive, because the network only sees cases in which White moves.

The representation scheme of Fig. 2(a) worked fairly well, but had one peculiar problem in that, after training, the network tended to prefer piling large numbers of men on certain points, in particular White's 5 point (the 20 point in the 1–24 numbering scheme). Figure 3 illustrates an example of this peculiar behavior. In this position White is to play 5–1. Most humans would play 4–5,4–9 in this position; however, the network chose the move 4–9,19–20. This is actually a bad move, because it reduces White's chances of making further points in his inner board. The fault lies not with the database used to train the network, but rather with the representation scheme used. In Fig. 2(a), notice that unit 13 is turned on whenever the final position is a point, and the number of men is different from the initial position. For the 20 point in particular, this unit will develop strong excitatory weights due to cases in which the initial position is not a point (i.e., the move makes the point). The 20 point

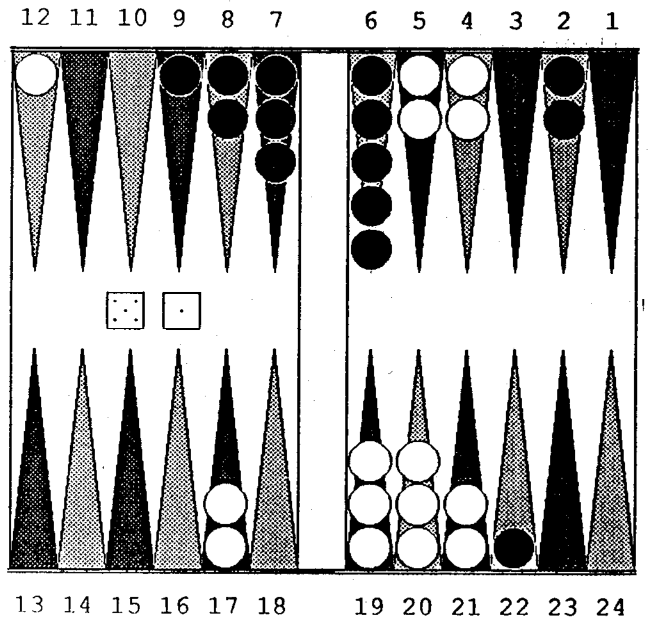

Fig. 3. A sample position illustrating a defect of the coding scheme of Fig. 2(a). White is to play 5–1. With the coding scheme of Fig. 2(a), the network prefers 4–9,19–20. With the coding scheme of Fig. 2(b), the network prefers 4–9,4–5.

is such a valuable point to make that the excitation produced by turning unit 13 on might overwhelm the inhibition produced by the poor distribution of builders.

In conceptual terms, humans would say that unit 13 participates in the representation of two different concepts: the concept of *making* a point, and the concept of *changing* the number of men occupying a made point. These two concepts are unrelated, and there is no point in representing them with a common input unit. A superior representation scheme in which these concepts are separated is shown in Fig. 2(b): In this representation unit 14 is turned on only for moves which make the point. Other moves which change the number of men on an already-made point do not activate unit 14, and thus do not receive any undeserved excitation. Similarly, units 12, 13 and 15 represent the important concepts of "slotting," "breaking," and "stripping" a point. With this representation scheme the network no longer tends to pile large numbers of men on certain points, and its overall performance is significantly better.

In addition to this representation of the raw board position, we also utilize a number of input units to represent certain "pre-computed" features of the raw input. The principal goal of this study has been to investigate network learning, rather than simply to obtain high performance, and thus we have resisted the temptation of including sophisticated hand-crafted features in the input encoding. However, we have found that a few simple features are needed in practice to obtain minimal standards of competent play. With only "raw" board information, the order of the desired computation (as defined by Minsky and Papert [29]) is undoubtedly quite high, since the computation of relevant features requires information to be integrated from over most of the board area. (See e.g. the discussion below concerning blot exposure.) The number of examples needed to learn such a difficult computation might be intractably large [14, 15, 21, 45]. By giving the network "hints" in the form of pre-computed features, this reduces the order of the computation, and thus might make more of the problem learnable in a tractable number of examples.

Our current coding scheme uses the following eight features: (1) pip count, (2) degree of contact, (3) number of points occupied in the inner board, (4) number of points occupied in the opponent's inner board, (5) total number of men in the opponent's inner board, (6) the presence of a "prime" formation, (7) blot exposure (the probability that a blot can be hit), and (8) strength of blockade (the probability that a man trapped behind an enemy blockade can escape). Readers familiar with the game will recognize that these are all conceptually simple features, which provide an elementary basis for describing board positions in a way that humans find relevant. However, the information provided by the features is still far removed from the actual computation of which move to make. Thus we are not "cheating" by explicitly giving the network information very close to the final answer in the input encoding. (We might add that we do not know how to "cheat" even if we wanted to.) The first

six features are trivial to calculate. The last two features, while conceptually simple, require a somewhat intricate sequential calculation. The blot exposure in particular, as described in the appendix, would be a very high-order Boolean predicate (a predicate order of 30 or more would not be surprising), and we suspect that the network might have trouble learning such computations. One should also note that the final two features involve real-valued probabilities between 0/36 and 36/36; thus we use analog coding in both the input and output layers.

An example of the importance of the pre-computed feature is presented in Fig. 4. In this position, White is to play 3–2 after Black started the game with a 5–1 by moving out to White's bar point. Without the blot exposure as a pre-computed feature, the network tends to play 12–15,12–14. This would be the correct move in the opening position, and undoubtedly the network's choice is due to the close similarity (in terms of Hamming distance) between this position and the opening position. In this position, however, the move is incorrect, because the blots on 14 and 15 are exposed to direct shots of 3 and 4. With the blot exposure explicitly calculated and presented to the network as a

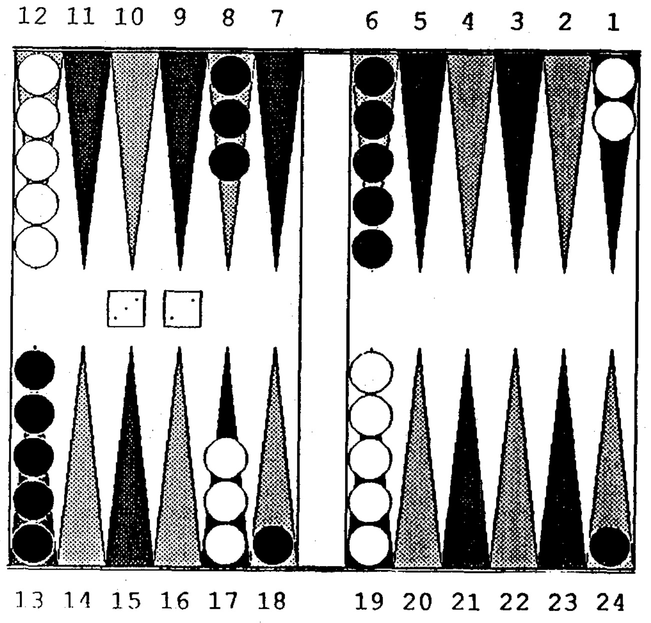

Fig. 4. A sample position illustrating the utility of pre-computed features in the input layer. White is to play 3–2. A network without features prefers 12–15,12–14 as in the opening. A network with a blot exposure feature prefers 12–14,17–20.

pre-computed feature, the network then prefers the move 12–14,17–20. This move surprised and impressed us: not only is it the best move by far in this situation, but we did not even think of it ourselves. There are two reasons why this is such a strong move: it gives excellent chances of making White's 5 point on the next turn, and it uses the principle of "duplication" to minimize the risk of a blot being hit. Note that the blot on 14 would be hit by a roll of 4, and the blot on 20 would also be hit by a roll of 4. Thus the total probability of at least one blot being hit is substantially smaller than other positions in which two different numbers hit the blots. It is quite interesting that the network acts as if it understands the principle of duplication; more likely, it probably is just trying to minimize the total blot exposure. (While not rigorously provable, this notion is supported by the large negative weights of the total blot exposure feature, and by the network's overall style of play.)

5. Training Procedure

To train the network, we have used the standard "back-propagation" learning algorithm [22, 32, 38, 39] for modifying the connections in a multilayer feed-forward network. Typical networks contained between 12 and 48 hidden units. Typical learning runs lasted for about 50 cycles through the training set, taking approximately 100–200 hours of CPU time on a Sun 3/160 with FPA. Following the notation of [41], typical learning parameter values were as follows: the momentum coefficient α was 0.9, the initial random weight scale was 0.1, the margin was 0.01, and the learning rate ε was 2.0 except for networks without hidden units, for which it was reduced to 0.5. As in the standard back-propagation training procedure, we cycle repeatedly through the database, presenting the patterns in order and changing the weights after every few presentations. However, our procedure differs in that it contains a few sources of noise which are not ordinarily present. The primary source of noise arises from the necessity of dealing with the large number of uncommented moves in the database. One solution would be simply to avoid presenting these moves to the network. However, this would limit the variety of input patterns presented to the network in training, and certain types of inputs probably would be eliminated completely. The alternative procedure which we have adopted is to skip the uncommented moves most of the time (75% for ordinary rolls and 92% for double rolls), and the remainder of the time present the pattern to the network and generate a random teacher signal with a slight negative bias. This makes sense, because if a move has not received a comment by the human expert, it is more likely to be a bad move than a good move. The random teacher signal is chosen uniformly from the interval [−65, +35]. The other source of noise is that we do not average over a fixed number of training patterns before updating the weights. Instead, the decision to update the weights is made randomly after every input with a 25% probability.

The sources of noise used in our training procedure are expected to give the network the benefit of experiencing a wider variety of input patterns during training, and to provide a mechanism for escaping from local minima. The potential disadvantage of this procedure is that the noise in the teacher signal could mislead the network about the actual value of certain patterns. To some extent, we expect this to be diminished by presenting the uncommented moves much less frequently than the commented moves. However, if the training continues for a very long time, the network might see the uncommented moves a sufficient number of times to draw false conclusions about their scores. We will discuss these issues in more quantitative terms below.

6. Measures of Performance

There are many different ways of assessing a network's performance once it has been trained. We have used the following four measures: (i) performance on the training data, (ii) performance on a set of test data which was not used to train the network, (iii) performance in actual game play against a conventional computer program (the program Gammontool of Sun Microsystems Inc., which is a competent intermediate-level program and the best commercial program that we have seen), and (iv) performance in game play against a human expert (G.T.). In the first two measures, we define the performance as the fraction of positions in which the network picks the correct move, i.e., those positions for which the move scored highest by the network agrees with the choice of the human expert. This is expected to be a more useful measure of performance than, for example, the fraction of patterns for which the network score is within a specified margin of the teacher score. In the latter two measures, the performance is defined simply as the fraction of games won,[2] without considering the complications of counting gammons or backgammons.

The first measure gives very little indication of the network's ability to generalize, and we shall not discuss it in detail. The remaining three measures each have their own merits. Performance against a human expert probably provides the most accurate assessment of the network's strength as a player, as well as an opportunity for detailed qualitative description of the network's play. However, this method is extremely time-consuming, and thus results for only a small number of games can be achieved. The understanding which arises through this method is primarily qualitative, and hence we postpone further discussion to Section 8. With automated game-playing against a conventional

[2] In the games against Gammontool, the network evaluation was used until the game became a pure race, at which point the Gammontool evaluation function was used for both sides. For the games against G.T., play proceeded until the point of either race or bear-off, after which best possible play (in the opinion of G.T.) was taken for both sides. These schema were necessary because the network received very little training on bear-off, and no training at all on racing situations.

program, more games can be played out, and accurate statistics can be obtained for a single, static network. However, if we wish to observe the performance over the entire time-course of a learning run, this method again is too time-consuming. Performance on a test set is the fastest testing method, but is not necessarily related to the network's actual game-playing strength.

In the discussion that follows, we examine questions such as how the network's performance depends on the number of hidden units, the number of hidden layers, the size of the training set, the noise in the training procedure, and the use of various pre-coded features in the input patterns. Some of these effects are discussed primarily in terms of performance on a test set, others are discussed in terms of performance versus Gammontool. In each case, the choice is based on the available data, and on the magnitude of the observed effect. It is interesting that some effects are more noticeable in the test set measure, while others are more noticeable in terms of number of games won. In each case, when results are presented only for one measure, we state qualitatively the results according to the other measure.

7. Quantitative Results

In Fig. 5, we present sample learning curves for a particular network as a function of the number of cycles through the training data, as measured by two indicators of performance: performance on the training data, and performance on a set of test data not used in training. (The training set was the 3202-position set described in Section 3, while the test set contained 1000 nonracing

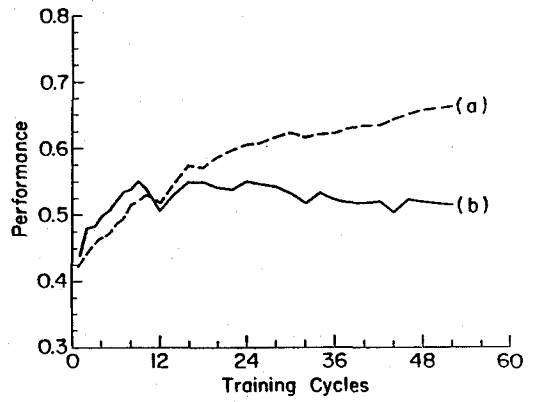

Fig. 5. Sample learning curves for a network trained on the 3200-position database with two layers of 24 hidden units each plotted versus number of cycles of training. (a) Performance on the training set (fraction of positions in which the network gives the highest score to the best move). (b) Performance on a test set (1000 positions) not used in training.

positions taken from games played by G.T. against Gammontool.) A number of important general points are illustrated in this figure. First, the learning of the training data appears to increase constantly as long as the training continues, with no sign of asymptoting. Secondly, performance on the test data does not continue to improve with more and more training. Instead, a maximum level of performance appears to be reached around 10–20 cycles through the training set, after which the performance either remains constant, or perhaps decreases slightly, with further training. This may be indicative of "overtraining" the network. Finally, the learning curves are not smooth; fluctuations are present. We expect that these fluctuations are due to our noisy training procedure, and do not indicate any repeatable effects.

7.1. Dependence on number of hidden units

In Fig. 6, we plot performance on the 1000-position test set versus number of cycles of training for networks with 0, 12, 24 and 48 hidden units in a single layer. Again we note that the networks do not always continue to improve with more and more training; for the networks with hidden units, a maximum performance generally appears to be reached somewhere around 20 cycles through the training set. The performance does generally increase as more hidden units are added, although the difference between the networks with 12 and 24 hidden units is very small and in fact the 12-hidden unit network seems to reach a slightly higher maximal performance. The most important point regarding this figure, however, is that the overall scale of variation is quite

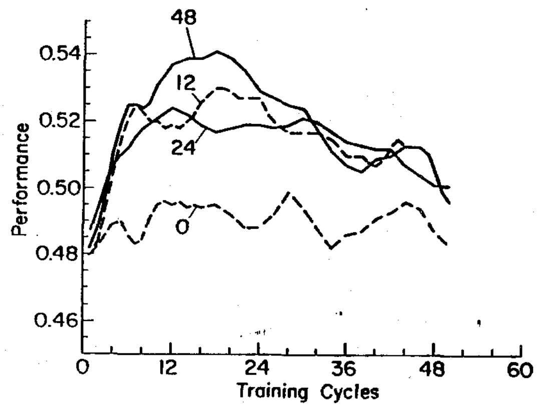

Fig. 6. Learning curves illustrating performance on the test set version cycles of training for networks with varying numbers of hidden units in a single layer, as indicated.

small. Adding more hidden units appears to give only a few percentage points improvement in performance, according to this measure. Even for the network without any hidden units, the performance is only about 5 percentage points lower than the optimal performance of the 48-hidden unit network. Similar results are found in terms of the percentage of games won against Gammontool, except that the network without hidden units appears to do noticeably worse according to this measure (about 8 to 10 percentage points worse than the 48-hidden unit network).

7.2. Dependence on number of hidden layers

Figure 7 shows the performance on the test set versus cycles of training for networks which have varying numbers of hidden layers. A few percentage point improvement is seen going from no hidden layers to a single layer, and a further smaller improvement is obtained with two hidden layers (which in this case have the same number of hidden units in each layer). We mention in passing, however, that a network with three layers of hidden units had lower performance than the one with two hidden layers, and in fact was slightly lower in performance than the network with a single hidden layer. Results in terms of games won show a similar behavior, except that there is a greater difference between the network without hidden units and the others, as described previously.

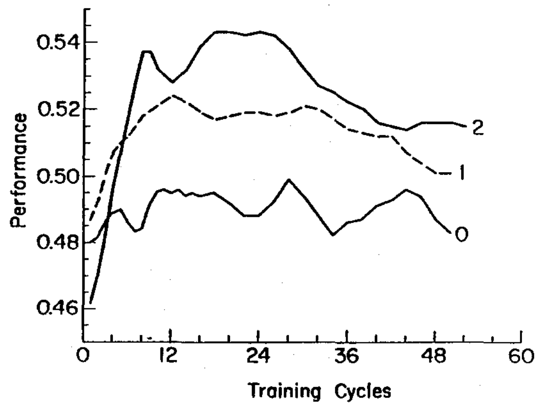

Fig. 7. Learning curves illustrating performance on the test set versus cycles of training for networks with varying numbers of layers of hidden units (24 units in each layer), as indicated. Maximal performance is obtained with two hidden layers. Performance of a network with three hidden layers falls slightly below the curve for a single layer.

7.3. Dependence on training set size

Figure 8 demonstrates the performance of three identical networks trained with training sets of different sizes. The smaller training sets were obtained by random selection of positions from the original training set. We find that the performance does decrease when a smaller training set is used, although once again the overall size of the effect is quite small. (The effect is also small in terms of game performance.) This provides evidence that simply adding more examples to the training set will not produce substantial improvements in performance if the choice of examples is not made intelligently. Instead, intelligent hand-crafting of particular examples to reinforce particular conceptual points is required.

We also mention in passing that if one simply observes the amount of training time needed to memorize the training set to a certain level of performance, one finds that the training time scales roughly quadratically with the size of the training set. This exponent is substantially larger than the value of 4/3 found for networks trained on 32-bit parity with a fixed training set [43], which was thought to be an upper bound on the value of this scaling exponent. The discrepancy might be due to the noise in the training procedure, or to the unusual measure of performance for this particular application.

7.4. Other effects

We have also examined the effects of eliminating the noise in the training procedure, and of changing the pre-computed features used in the input coding scheme. These effects are much larger than the effects discussed so far. The

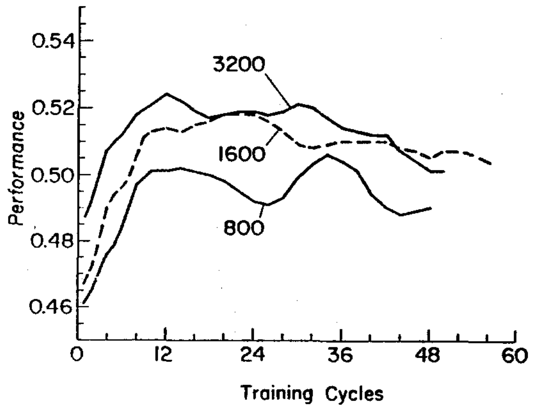

Fig. 8. Learning curves illustrating performance on the test set versus cycles of training for networks trained with different sized training sets (number of positions as indicated).

most dramatic effect is seen by removing entirely all the precomputed features from the input coding, and using instead only "raw" information describing the board position and move. This produces a dramatic decrease of around 15 to 20 percentage points in the number of games won by the network against Gammontool, although the performance on the test set is not affected so significantly. Eliminating the noise also has a large effect: when the uncommented moves in the training set are never seen by the network, the percentage of games won against Gammontool falls by approximately 10 to 15 percentage points. (Removal of the random score noise and leaving the other sources of randomness produces a less severe performance drop of only 7–8 percentage points.) Once again, the performance on the test set does not drop so much (this measure changes by only about 5 percentage points).

A summary of these various effects as measured by performance against Gammontool is presented in Table 2. The best network that we have produced so far appears to defeat Gammontool nearly 60% of the time. Using this as a benchmark, we find that the most serious decrease in performance occurs by removing all pre-computed features from the input coding. This produces a network which wins at most about 41% of the time. The next most important effect is the removal of noise from the training procedure; this results in a network which wins 45% of the time. Next in importance is the presence of hidden units; a network without hidden units wins about 50% of the games against Gammontool. In contrast, the other effects we have discussed, such as varying the exact number of hidden units, the number of layers, or the size of the training set, results in only a few (1–3) percentage point decrease in the number of games won.

Table 2
Summary of performance statistics for various networks. (a) The best network we have produced, containing two layers of hidden units, with 24 units in each layer. (b) A network with only one layer of 24 hidden units. (c) A network with 24 hidden units in a single layer, trained on a training set half the normal size. (d) A network with half the number of hidden units as in (b). (e) A network with features from the Gammontool evaluation function substituted for the normal features. (f) A network without hidden units. (g) A network trained with no noise in the training procedure. (h) A network with only a raw board description as input.

Network size	Training cycles	Performance on test set	Performance vs. Gammontool	Comments
(a) 459-24-24-1	20	0.540	0.59 ± 0.03	
(b) 459-24-1	22	0.542	0.57 ± 0.05	
(c) 459-24-1	24	0.518	0.58 ± 0.05	1600-position database
(d) 459-12-1	10	0.538	0.54 ± 0.05	
(e) 410-24-12-1	16	0.493	0.54 ± 0.03	Gammontool features
(f) 459-1	22	0.485	0.50 ± 0.03	No hidden units
(g) 459-24-12-1	10	0.499	0.45 ± 0.03	No training noise
(h) 393-24-12-1	12	0.488	0.41 ± 0.02	No features

Also included in Table 2 is the result of an interesting experiment in which we removed our usual set of pre-computed features and substituted instead the individual terms of the Gammontool evaluation function. We found that the resulting network, after being trained on our expert training set, was able to defeat the Gammontool program by a small margin of 54 to 46 percent. The purpose of this experiment was to provide evidence of the usefulness of network learning as an adjunct to standard AI techniques for hand-crafting evaluation functions. Given a set of features to be used in an evaluation function which have been designed, for example, by interviewing a human expert, the problem remains as to how to "tune" these features, i.e., the relative weightings to associate to each feature, and at an advanced level, the context in which each feature is relevant. Little is known in general about how to approach this problem, and often the human programmer must resort to painstaking trial-and-error tuning by hand. We claim that network learning is a powerful, general-purpose, automated method of approaching this problem, and has the potential to produce a tuning which is superior to those produced by humans, given a database of sufficiently high quality, and a suitable scheme for encoding the features. The result of our experiment provides evidence to support this claim, although it is not firmly established since we do not have highly accurate statistics, and we do not know how much human effort went into the tuning of the Gammontool evaluation function. More conclusive evidence would be provided if the experiment were repeated with a more sophisticated program such as Berliner's BKG and similar results were obtained.

8. Qualitative Results

Analysis of the weights produced by training a network is an exceedingly difficult problem, which we have only been able to approach qualitatively. In Fig. 9 we present a diagram showing the connection strengths in a network with 651 input units and no hidden units. The figure shows the weights from each input unit to the output unit. (For purposes of illustration, we have shown a coding scheme with more units than normal to explicitly represent the transition from initial to final position.) Since the weights go directly to the output, the corresponding input units can be clearly interpreted as having either an overall excitatory or inhibitory effect on the score produced by the network.

A great deal of columnar structure is apparent in Fig. 9. This indicates that the network has learned that a particular number of men at a given location, or a particular type of transition at a given location, is either good or bad independent of the exact location on the board where it occurs. For example, column L encodes transitions which make points. The large white squares appearing in this column indicate that the network has discovered the general

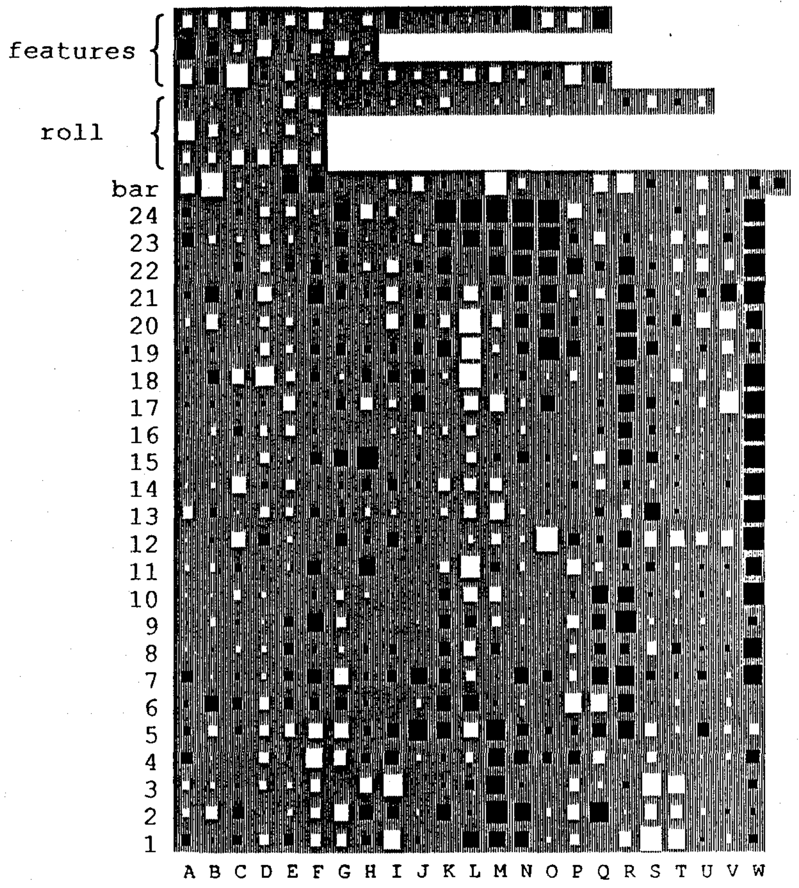

Fig. 9. A Hinton diagram for a network with 651 input units and no hidden units. Small squares indicate weights from a particular input unit to the output unit. White squares indicate positive weights, and black squares indicate negative weights. Size of square indicates magnitude of weight. First 24 rows from bottom up indicate raw board information. Letting x be the number of men before the move and y the number of men after the move, the interpretations of columns are as follows: A: $x \leqslant -5$; B: $x = -4$; C: $x = -3$; D: $x \leqslant -2$; E: $x = -1$; F: $x = 1$; G: $x = 2$; H: $x = 3$; I: $x = 4$; J: $x \geqslant 5$; K: $x < 1$ & $y = 1$; L: $x < 2$ & $y \geqslant 2$; M: $x < 3$ & $y = 3$; N: $x < 4$ & $y = 4$; O: $x < y$ & $y \geqslant 5$; P: $x = 1$ & $y = 0$; Q: $x \geqslant 2$ & $y = 0$; R: $x \geqslant 2$ & $y = 1$; S: $x \geqslant 3$ & $y = 2$; T: $x \geqslant 4$ & $y = 3$; U: $x \geqslant 5$ & $y = 4$; V: $x > y$ & $y \geqslant 5$; W: probability of a White blot at this location being hit (pre-computed feature). The next row encodes the number of men on White and Black bars. The next 3 rows encode roll information. Remaining rows encode various pre-computed features.

utility of making points. Similarly, column R encodes transitions which break points, and the network has found that this is generally a bad idea. Columns M, N and O represent "piling" transitions which increase the number of men on existing points, and the network has learned that this is bad for points in the opponent's inner board, and for advanced points in the network's inner board. Columns S and T encode "unpiling" transitions, and the network gives high scores to moves which unpile the points held in the opponent's inner board. Finally, column W represents the pre-computed feature of blot exposure. The network has clearly learned the general principle that moves which leave blots exposed are bad moves.

The upper part of Fig. 9 contains the encoding of the bar and dice roll information, as well as the rest of the pre-computed features. We can also make some observations on which of these input units the network finds significant. The most important features seem to be the number of points held in the network's inner board, and the total blot exposure.

Much insight into the basis for the network judgment of various moves has been gained by actually playing games against it. In fact, one of the most revealing tests of what the network has and has not learned came from a 20-game match played by the G.T. against one of the latest generation of networks with 48 hidden units. (A detailed description of the match is given in [44].) The surprising result of this match was that the network actually won, 11 games to 9. However, a detailed analysis of the moves played by the network during the match indicates that the network was extremely lucky to have won so many games, and could not reasonably be expected to continue to do so well over a large number of games. Out of the 20 games played, there were 11 in which the network did not make any serious mistakes. The network won 6 out of these 11 games, a result which is quite reasonable. However, in 9 of the 20 games, the network made one or more serious (i.e. potentially fatal) "blunders." The seriousness of these mistakes would be equivalent to dropping a piece in chess. Such a mistake is nearly always fatal in chess against a good opponent; however in backgammon there are still chances due to the element of luck involved. In the 9 games in which the network blundered, it did manage to survive and win 5 of the games due to the element of luck. (We are assuming that the mistakes made by the human, if any, were only minor mistakes.) It is highly unlikely that this sort of result would be repeated. A much more likely result would be that the network would win only one or two of the games in which it made a serious error. This would put the network's expected performance against expert or near-expert humans at about the 35–40% level. (This has also been confirmed in play against other networks.)

As for the specific kinds of mistakes made by the network, we find that they are not at all random, senseless mistakes, but instead fall into clear, well-defined conceptual categories, and furthermore, one can understand the reasons why these categories of mistakes are made. The most frequent kinds of major mistakes made by the network are: (i) mistakes made while bearing off against opposition, or opposing the opponent's bear-off, (ii) mistakes involving the play of outfield blots (for example, failing to move them to safety past the opponent's midpoint), (iii) failing to escape from the opponent's blockade while the opponent is closed out or trapped behind a prime, (iv) clearing or breaking its 6 point in totally inappropriate situations, and (v) inappropriate attempts to blitz, and failure to blitz in some situations where it is called for. In each case, we believe we understand why the network behaves as it does. In the case of problem (i), this undoubtedly occurs simply because we have not made a systematic effort to include these kinds of positions in the database. For problem (ii), the network has much less experience with outfield play in general, because the outfield locations are occupied much less frequently than the infield locations. For problem (iii), the network has been shown in many other situations that it should not escape the blockade, either because it is behind and needs to wait for a hit, or it is too dangerous to escape, or simply that there is no legal move which escapes the blockade. What the network must learn is the proper context in which blockade escapes should be made. For problem (iv), the network again has been told many times that it is often a good idea to break or clear the 6 point, and once again it is a matter of learning the right context. For problem (v), the failure to carry out a blitz where it is called for seems to be due to the existence of some other alternative, such as making a side prime, which is ordinarily quite valuable, but not as valuable as the blitz attack. Inappropriate attempts to blitz probably result once again from a failure to discern the context in which a blitz attack is desirable. In each case, we are confident that these particular problems could be remedied by including a large number (i.e. on the order of a few dozen) examples of positions of each characteristic type in the data base used to train the network.

Of course, it is also worth mentioning the kinds of situations that the network handles correctly. We find that the network does act as if it has picked up many of the global concepts and strategies of advanced play. The network appears to have learned the general principles of a running game strategy, and it usually implements this strategy quite effectively. The network also plays a holding game strategy reasonably well (in part thanks to the hand-crafted examples discussed in Section 3 teaching it the value of hitting), although it has trouble in filling in its inner board while waiting in a technically precise fashion. The network has learned the general usefulness of making a prime formation, although it apparently has not learned many of the long-term consequences of such a formation. Also the elements of a blitz attack have been learned quite well, and in the appropriate circumstances the network is capable of blitzing very effectively. The back game strategy is probably the most difficult game strategy to learn, for humans as well as for the network. Previous generations of the network were not able to capture any of the ideas of how and when to

play a back game, or to defend against a back game. The current networks, while they usually do not pick the exact best move, seem to have at least caught on to some of the general operating principles in these situations. For example, in about half the situations in which the network has an oppurtunity to hit, it will choose not to do so, indicating that it recognizes this ingredient of a back game strategy. As such, this probably puts the network well beyond the capabilities of nearly all conventional programs, which usually have no idea at all how to play a back game, and in fact make precisely the wrong move nearly all the time.

In addition to elements of global strategies of play, the network has also learned many important tactical elements of play at the advanced level. The network has learned the value of slotting, particularly on its 5 point, although it does not know perfectly when slotting is and is not appropriate. The battle for the 5 points which often occurs early in expert games has been learned very well by the network, and it can execute this tactic nearly as well as humans. The network appears to understand the "action play," a type of move which involves increasing board coverage when the opponent has only one man back, although it does not understand very well the notion of coverage in general. Other tactics such as duplication of blot exposure, and hitting and splitting, have also been learned by the network.

To summarize, qualitative analysis of the network's play indicates that it has learned many important strategies and tactics of advanced backgammon. This gives the network very good overall performance in typical positions. However, the network's worst case performance leaves a great deal to be desired. The network is capable of making both serious, obvious "blunders," as well more subtle mistakes, in many different types of positions. Worst case performance is important, because the network must make long sequences of moves throughout the course of a game without any serious mistakes in order to have a reasonable chance of winning against a skilled opponent. the prospects for improving the network's worst case performance appear to be mixed. It seems quite likely that many of the current "blunders" can be fixed with a reasonable number of hand-crafted examples added to the training set. However, many of the subtle mistakes are due to a lack of very sophisticated knowledge, such as the notion of timing. It is difficult to imagine that this kind of knowledge could be imparted to the network in only a few examples. Probably what is required is either an intractably large number of examples, or a major overhaul in either the pre-computed features or the training paradigm.

9. Discussion

We have seen from both quantitative and qualitative measures that the network has learned a great deal about the general principles of backgammon play, and has not simply memorized the individual positions in the training set.

Quantitatively, the measure of game performance provides a clear indication of the network's ability to generalize, because apart from the first couple of moves at the start of each game, the network must operate entirely on generalization. Qualitatively, one can see after playing several games against the network that there are certain characteristic kinds of positions in which it does well, and other kinds of positions in which it systematically makes well-defined types of mistakes. The most concise summary of the network's style of play, in our opinion, is that it seems to make the "intuitively obvious" play, from the human expert's point of view. In other words, much of the knowledge learned by the network has the character of what a human would call common sense. As a result, the network is capable of doing certain things which are extremely difficult to obtain from conventional programs.

A good example of this is provided in Fig. 10, which illustrates a position from the match between Berliner's BKG program and the human world backgammon champion. In this position White can either play 20–24,21–23, leaving a blot on the 20 point, or 16–20,16–18, leaving a blot on the 18 point. The first play has a slightly lower immediate risk of being hit (11/36 versus

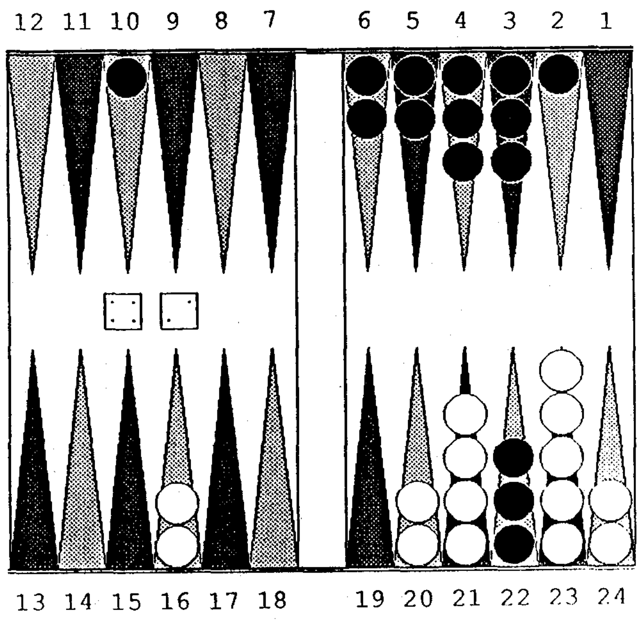

Fig. 10. A sample position from [4] involving considerations of long-term versus short-term risk. White to play 4–2. The correct move is 16–18,16–20. Conventional computer programs typically pick 20–24,21–23. The network selects the right move by a clear margin.

13/36), but since the two men remain on the 16 point, there is still a long-term risk of eventually having to leave another man open. The second play is correct, because it does away with the long-term risk entirely. To make this type of play with a conventional evaluation function, it is necessary to devise a set of features which compute both the short-term and long-term risks, and which recognize the characteristic kinds of situations in which these computations are relevant. This goes well beyond the sophistication of commercial programs, which usually make the wrong play here. (Berliner's program did make the correct play, because its evaluation function does have the required sophistication.) On the other hand, the network has no trouble finding the right move, despite the fact that it is explicitly given the short-term risk as a pre-computed feature, but is not told about the long-term risk.

How is the network able to achieve this? We suggest that what the network learns from the many examples in the data base are the expert's general tendencies to make certain kinds of moves in various kinds of situations. The expert would describe these tendencies as being intuitively appealing or obvious, and in this particular case, the intuitively obvious move happens to be the correct one. Berliner himself says "This play is certainly the correct one, which any expert should make *without thinking* (emphasis added), but it is not the kind of thing computer programs are supposed to be able to do." [4] From our experience with the network, it is usually the case that, if a human expert can select the right move *without thinking*, then the network will also be able to find the right move.

Or course, this is not to say that the network has fully captured the expert's intuition, or that its level of play approaches expert level. Due to the network's frequent "blunders," its overall level of play is only intermediate level, although it probably is somewhat better than the average intermediate-level player. Against the intermediate-level program Gammontool, our best network wins almost 60% of the games. However, against a human expert the network would only win about 35–40% of the time.

The network's level of play is all the more impressive when one considers that our simple supervised learning approach leaves out some very important sources of information which are readily available to humans. The network is never told that the underlying topological structure of its input space really corresponds to a one-dimensional spatial structure; all it knows is that the inputs form a 459-dimensional hypercube. (The notions of spatial relationships between different board locations, e.g. the notion of "coverage," would be particularly useful to the network.) It has no idea of the object of the game, nor of the sense of temporal causality, i.e. the notion that its actions have consequences, and how those consequences lead to the achievement of the objective. The teacher signal only says whether a given move is good or bad, without giving any indication as to what the teacher's reasons are for making such a judgment. Finally, the network is only capable of scoring single moves in isolation, without any idea of what other moves are available. These sources of knowledge are essential to the ability of humans to play backgammon well, and it seems likely that some way of incorporating them into the network learning paradigm will be necessary in order to achieve further substantial improvements in performance.

There are a number of ways in which these additional sources of knowledge might be incorporated, and we shall be exploring some of them in future work. For example, knowledge of the underlying 1-D spatial structure could be imparted by propagating a fraction of a given weight change to spatially neighboring weights; this would tend to enforce a kind of approximate translation invariance. Knowledge of alternative moves could be introduced by defining a more sophisticated error signal which takes into account not only the network and teacher scores for the current move, but also the network and teacher scores for other moves from the same position. However, the more immediate plans involve a continuation of the existing strategies of hand-crafting examples and coding scheme modifications to eliminate the most serious errors in the network's play. If these errors can be eliminated, and we are confident that this can be achieved, then the network would become substantially better than any commercially available program, and would be a serious challenge for human experts. We would expect 65% performance against Gammontool, and 45% performance against human experts.

Some of the results of our study have implications beyond backgammon to more general classes of difficult problems. It has seemed possible to some researchers that connectionist learning procedures would make it possible to effortlessly construct intelligent artificial systems. According to this view, a simple coding scheme for the inputs and outputs, and a random database of sample input–output pairs, would suffice to produce a network capable of producing correct outputs in general for all inputs. However, our work indicates that this is not likely to hold for general problems for two important reasons. First, some coding schemes are better than others, because certain aspects of a coding scheme may lead it to produce false generalizations which might only be corrected with an excessive number of training patterns. A great deal of effort must be spent in developing appropriate schemes for encoding the input–output information. This must be done largely on a trial and error basis, given the general absence of theoretical principles for choosing good coding schemes. (Our suggestion of using "conceptually significant" coding schemes, i.e., choosing the individual bits of the coding scheme to correspond to concepts that humans actually use in the domain, may turn out to be an important exception.) Secondly, randomly chosen training patterns might not work, because certain conceptual principles of the given domain might be over-represented in the sample, and other conceptual principles might be under-represented. In the worst case, it will be necessary to intelligently design the training set, using an intelligent mechanism for reducing the number of

over-represented situations, and an intelligent mechanism for designing additional patterns to illustrate the under-represented situations. Again a substantial amount of human labor is required to carry this out. A flow chart summary of this process is illustrated in Fig. 11.

An additional problem which might occur is that even with a good coding scheme and a good training set, the minimal training time and training set size would scale exponentially with the order of computation being trained [14, 15, 21, 45], and thus high-order computations would effectively be unlearnable. This motivates the search for novel learning algorithms and procedures which

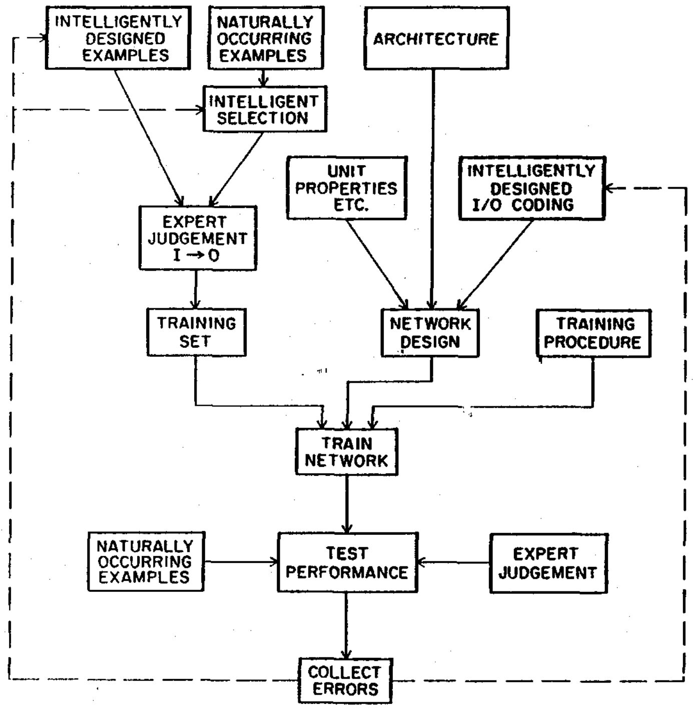

Fig. 11. Flow chart indicating general process of developing an artificial intelligent system using connectionist learning. The three main components of network development are: construction of the training set, network design, and choice of training procedure. After training, one can test the network's performance, and accumulate errors which can be used to improve performance. In general, the process cannot be fully automated. Intelligent design of the training examples is necessary, both to reduce representation of common positions, and to hand-craft examples illustrating rare but important positions. Also intelligent design of the coding scheme using domain-specific knowledge is necessary. Ideas for training examples and coding schemes come from observing the kinds of errors made by the network after training.

would scale only polynomially with the order of computation. Such algorithms would clearly be unable to learn completely arbitrary Boolean functions, but the kinds of functions one encounters in typical real-world problems are far from arbitrary, and it might be possible to include enough additional information about the structure of the problem so that it could be learned with a tractable number of examples.

On the positive side, we foresee a potential for using these networks as a novel type of expert system. In typical commercial applications, the expert system serves as an automated assistant, which handles most of the routine inputs normally analyzed by a human expert. Occasionally there will be an input which is too difficult for the system to handle. In that case, the system should be able to recognize the difficulty and notify the human expert that human attention is required. It might be possible to implement a similar feature in connectionist expert systems. For our particular application, a simple way to do this would be to send the "difficulty" signal in the event that the network is not able to find a clear best move, i.e., the difference in score between the network's chosen move and the competing alternatives is quite close. A more intriguing possibility is to define an additional teacher signal for an additional output unit as follows: for each position in the training set, if the network finds the right move (i.e. if the move it scores the highest was scored +100 by the human expert), then the teacher signal is 1, else 0. Assuming that the network is able to learn this input–output function reasonably well, the second output bit would serve as a sort of "confidence indicator" indicating to what extent the network thinks it is capable of finding the right move in a given position.

We also foresee a potential for combining connectionist learning techniques with conventional AI techniques for hand-crafting knowledge to make significant progress in the development of intelligent systems. From the practical point of view, network learning can be viewed as an "enhancer" of traditional techniques, which might produce systems with superior performance. For this particular application, the obvious way to combine the two approaches is in the use of pre-computed features in the input encoding. Any set of hand-crafted features used in a conventional evaluation function could be encoded as discrete or continuous activity levels of input units which represent the current board state along with the units representing the raw information. Given a suitable encoding scheme for these features, and a training set of sufficient size and quality (i.e., the scores in the training set should be better than those of the original evaluation function), it seems possible that the resulting network could outperform the original evaluation function, as evidenced by our experiment with the Gammontool features.

Finally, network learning might also hold promise as a means of achieving the long-sought goal of automated feature discovery, as evidenced by recent applications in signal processing [13] as well as symbolic information processing

[39]. Our network certainly appears to have learned a great deal of knowledge from the training set which goes far beyond the amount of knowledge that was explicitly encoded in the input features. Some of this knowledge (primarily the lowest-level components) is apparent from the weight diagram when there are no hidden units (Fig. 9). However, much of the network's knowledge remains inaccessible. What is needed now is a means of disentangling the novel features discovered by the network from either the patterns of activity in the hidden units, or from the massive number of connection strengths which characterize the network. This is one of our top priorities for future research, although techniques for such "reverse engineering" of parallel networks are only beginning to be developed [34, 35, 41].

Appendix A. Glossary of Backgammon Terms

– *Action play.* A tactic involving splitting the back men in an attempt to gain additional chances to hit the opponent. This usually occurs after one or both of the opponent's back men has escaped from behind one's blockade.

– *Back game.* An extreme version of the holding game strategy in which two or more points are occupied in the opponent's inner board. This is probably the most difficult strategy to learn. In this situation, many of the normal strategies of play are exactly reversed. For example, one often passes up chances to hit the opponent so as not to impede his forward progress.

– *Backgammon.* Similar to gammon except that there is a further requirement that the opponent has one or more men in the opponent's inner board. The number of points won is triple the normal value.

– *Bar.* The strip in the middle of the board where blots are placed after they are hit. From here, men must re-enter in the opponent's inner board, so men on the bar are effectively all the way back at the start in terms of the race to the finish. All men must be moved off the bar before other men can be moved. If a man on the bar cannot be moved, the player loses his turn.

– *Bear-off.* The final stage of the game, in which one moves one's pieces off the board. This can only take place when all men are in one's inner board.

– *Blitz game.* A strategy consisting of an all-out attack attempting to close out the opponent (i.e. trap men on the bar behind a completely closed inner board, so that no movement is possible) and win a gammon. In this strategy, greater than normal risks are taken. For example, one often hits the opponent and leaves a man in the inner board exposed to a return hit.

– *Blockade.* A configuration of several nearby points which restricts partially or completely the opponent's ability to move any pieces located behind the blockade.

– *Blot.* A single man at a particular location. An opponent piece can land on this location. When this happens, the blot is said to have been "hit" and is moved to the bar.

– *Blot exposure.* For a given blot, the number of rolls out of 36 which would allow the opponent to hit the blot. The total blot exposure is the number of rolls out of 36 which would allow the opponent to hit any blot. Blot exposure depends on: (a) the locations of all enemy men in front of the blot; (b) the number and location of blocking points between the blot and the enemy men; and (c) the number of enemy men on the bar, and the rolls which allow them to re-enter the board, since men on the bar must re-enter before blots can be hit.

– *Breaking*. A move in which you initially have two or more men at a location, and end up with only one man present.

– *Builder*. Usually refers to the extra men on a point which can be moved to make new points without breaking the existing one.

– *Clearing*. A move in which you initially have two or more men at a given location, and end up with no men present.

– *Contact*. A situation in which the opposing forces are engaged, i.e., each side has to move past a certain number of the opponent's pieces in order to reach the finish. Hitting and blocking are possible in such situations.

– *Gammon*. When one side removes all pieces from the board before the opponent has removed a single piece, this is called winning a gammon. The number of points won is double the regular value.

– *Holding game*. A strategy in which two or more men are deliberately left on a point far back waiting for a chance to hit the opponent.

– *Inner board*. Locations 1–6 constitute Black's inner board, while locations 19–24 constitute White's inner board.

– *Outer board*. Locations 7–12 constitute Black's outer board, while locations 13–18 constitute White's outer board.

– *Outfield*. A term referring the entire outer board region (locations 7–18).

– *Piling*. Placing too many men (usually four or more) on a single point.

– *Point*. When one side has two or more men on a particular board location, this is referred to as having established, or "made" that point. Opponent pieces then cannot land on this location.

– *Priming game*. A strategy which involves trapping the opponent behind a prime, i.e., a blockade consisting of six consecutive points. Such a blockade is impossible to escape.

– *Race*. A situation in which no contact is present.

– *Running game*. A strategy which consists primarily of attempting to escape the opponent's blockade and move one's men safely home, while at the same time hitting the opponent and trapping him behind one's own blockade.

– *Slotting*. An expert tactic of deliberately leaving a blot exposed to a direct hit, in the hopes of covering it and making a point on the next run.

– *Stripping*. A move which removes all builders from a point and leaves only two men there. Any further moves from this point will break it.

ACKNOWLEDGMENT

This work was inspired by a conference on "Evolution, Games and Learning" held at Los Alamos National Laboratory, May 20-24, 1985. We thank Hans Berliner for providing some of the positions used in the database, Sun Microsystems Inc. for providing the source code for their Gammontool program, Subutai Ahmad for writing the weight display graphics package, Bill Bogstad for assistance in programming the back-propagation simulator, and Bartlett Mel, Peter Frey and Scott Kirkpatrick for critical reviews of the manuscript. G.T. was supported in part by the National Center for Supercomputing Applications. T.J.S. was supported by a NSF Presidential Young Investigator Award, and by grants from the Seaver Institute and the Lounsbury Foundation.

REFERENCES

1. Ackley, D.H. and Berliner, H.J., The QBKG system: Knowledge representation for producing and explaining judgments, Tech. Rept. CMU-CS-83-116, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA (1983).
2. Ackley, D.H., Hinton, G.E. and Sejnowski, T.J., A learning algorithm for Boltzmann machines, *Cognitive Sci.* 9 (1985) 147–169.
3. Bachrach, J.R., Connectionist learning in backgammon, COINS Tech. Rept., University of Massachusetts, Amherst, MA (1986).
4. Berliner, H., Computer backgammon, *Sci. Am.* 243 (1) (1980) 64–72.
5. Berliner, H., Experiences in evaluation with BKG: A program that plays backgammon, in: *Proceedings IJCAI-77*, Cambridge, MA (1977) 428–433.
6. Berliner, H., On the construction of evaluation functions for large domains, in: *Proceedings IJCAI-79*, Tokyo (1979) 53–55.
7. Bourland, H. and Wellekens, C.J., Speech pattern discrimination and multilayer perceptrons, Tech. Rept. M.211, Philips Research Laboratory, Brussels, Belgium (1987).
8. Cooke, B., *Championship Backgammon* (Prentice-Hall, Englewood Cliffs, NJ, 1979).
9. Cooke, B. and Bradshaw, J., *Backgammon: The Cruelest Game* (Random House, New York, 1974).
10. Dwek, J., *Backgammon for Profit* (Stein and Day, New York, 1978).
11. Frey, P.W. (Ed.), *Chess Skill in Man and Machine* (Springer, New York, 2nd ed., 1983).
12. Frey, P.W., Algorithmic strategies for improving the performance of game-playing programs, in: D. Farmer, A. Lapedes, N. Packard and B. Wendroff (Eds.), *Evolution, Games and Learning* (North-Holland, Amsterdam, 1986).
13. Gorman, R.P. and Sejnowski, T.J., Analysis of hidden units in a layered network trained to classify sonar targets, *Neutral Networks* 1 (1988) 75–89.
14. Hampson, S.E. and Volper, D.J., Linear function neurons: Structure and training, *Biol. Cybern.* 53 (1986) 203–217.
15. Hampson, S.E. and Volper, D.J., Disjunctive models of Boolean category learning, *Biol. Cybern.* 56 (1987) 121–137.
16. Hinton, G.E., Connectionist learning procedures, Tech. Rept. CMU-CS-87-115, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA (1987).
17. Hinton, G.E. and Sejnowski, T.J., Learning and relearning in Boltzmann machines, in: J.L. McClelland and D.E. Rumelhart (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* 2 (MIT Press, Cambridge, MA, 1986).
18. Holland, J.H., Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems, in: R.S. Michalski, J. Carbonell and T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach* II (Morgan Kaufmann, Los Altos, CA, 1986).
19. Holland, T., *Backgammon for People Who Hate to Lose* (David McKay, New York, 1977).
20. Jacoby, O. and Crawford, J.R., *The Backgammon Book* (Bantam Books, New York, 1970).
21. Judd, J.S., Complexity of connectionist learning with various node functions, COINS Tech. Rept. 87-60, University of Massachusetts, Amherst, MA (1987).
22. Le Cun, Y., A learning procedure for asymmetric network, in: *Proceedings of Cognitiva (Paris)* 85 (1985) 599–604.

23. Le Cun, Y., Modèles connectionistes de l'Apprentissage, Ph.D. Thesis, University of Paris VI (1987).
24. Lehky, S. and Sejnowski, T.J., Network model of shape-from-shading: Neural function arises from both receptive and projective fields, *Nature* 333 (1988) 452–454.
25. Levy, D.N.L. (Ed.), *Computer Gamesmanship: The Complete Guide to Creating and Structuring Intelligent Game Programs* (Simon and Schuster, New York, 1983).
26. Magriel, P., *Backgammon* (Times Books, New York, 1976).
27. Maxwell, T., Giles, C.L. and Lee, Y.C., Generalization in neural networks: the contiguity problem, in: *Proceedings IEEE International Conference on Neural Networks* (1987).
28. Michalski, R.S., Carbonell, J.G. and Mitchell, T.M. (Eds.), *Machine Learning: An Artificial Intelligence Approach* (Tioga, Palo Alto, CA, 1983).
29. Minsky, M. and Papert, S., *Perceptrons* (MIT Press, Cambridge, MA, 1969).
30. Mitchell, D.H., Using features to evaluate positions in expert's and novices' Othello games, Master's Thesis, Northwestern University, Evanston, IL (1984).
31. Qian, N. and Sejnowski, T.J., Predicting the secondary structure of globular proteins using neural network models, *J. Mol. Biol.* 202 (1988) 865–884.
32. Parker, D.B., Learning-logic, Tech. Rept. TR-47, MIT Center for Computational Research in Economics and Management Science, Cambridge, MA (1985).
33. Prager, R.W., Harrison, T.D. and Fallside, F., Boltzmann machines for speech recognition, *Comput. Speech Lang.* 1 (1987) 3–27.
34. Rosenberg, C.R., Revealing the structure of NETtalk's internal representations, in: *Proceedings Ninth Annual Conference of the Cognitive Science Society* (Erlbaum, Hillsdale, NJ, 1987).
35. Rosenberg, C.R., Learning the connection between spelling and sound: A network model of oral reading, Ph.D. Thesis, Princeton University, Princeton, NJ (1987).
36. Rosenblatt, F., *Principles of Neurodynamics* (Spartan Books, New York, 1959).
37. Rumelhart, D.E. and McClelland, J.L. (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* 1–2 (MIT Press, Cambridge, MA, 1986).
38. Rumelhart, D.E., Hinton, G.E. and Williams, R.J., Learning internal representations by error propagation, in: D.E. Rumelhart and J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* 1 (MIT Press, Cambridge, MA, 1986).
39. Rumelhart, D.E. Hinton, G.E. and Williams, R.J., Learning representations by back-propagating errors, *Nature* 323 (1986) 533–536.
40. Samuel, A.L., Some studies in machine learning using the game of checkers, *IBM J. Res. Dev.* 3 (1959) 210–229.
41. Sejnowski, T.J. and Rosenberg, C.R., Parallel networks that learn to pronounce English text, *Complex Syst.* 1 (1987) 145–168.
42. Sutton, R.S., Learning to predict by the methods of temporal differences, GTE Labs Tech. Rept. TR87-509.1 (1987).
43. Tesauro, G., Scaling relationships in back-propagation learning: dependence on training set size, *Complex Syst.* 1 (1987) 367–372.
44. Tesauro, G., Neural network defeats creator in backgammon match, Tech. Rept. No. CCSR-88-6, Center for Complex Systems Research, University of Illinois, Urbana-Champaign, IL (1988).
45. Tesauro, G. and Janssens, B., Scaling relationships in back-propagation learning: dependence on predicate order, *Complex Syst.* 2 (1988) 39–44.
46. Widrow, B. and Hoff, M.E., Adaptive switching circuits, *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record* 4 (1960) 96–104.

Received October 1987; revised version received June 1988