Machine Learning, 8, 257-277 (1992)

Practical Issues in Temporal Difference Learning

GERALD TESAURO
IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 USA

Abstract. This paper examines whether temporal difference methods for training connectionist networks, such as Sutton's TD(λ) algorithm, can be successfully applied to complex real-world problems. A number of important practical issues are identified and discussed from a general theoretical perspective. These practical issues are then examined in the context of a case study in which TD(λ) is applied to learning the game of backgammon from the outcome of self-play. This is apparently the first application of this algorithm to a complex non-trivial task. It is found that, with zero knowledge built in, the network is able to learn from scratch to play the entire game at a fairly strong intermediate level of performance, which is clearly better than conventional commercial programs, and which in fact surpasses comparable networks trained on a massive human expert data set. This indicates that TD learning may work better in practice than one would expect based on current theory, and it suggests that further analysis of TD methods, as well as applications in other complex domains, may be worth investigating.

Keywords. Temporal difference learning, neural networks, connectionist methods, backgammon, games, feature discovery

1. Introduction

One of the most fascinating and challenging paradigms of traditional machine learning research is the *delayed reinforcement* learning paradigm. In the simplest form of this paradigm, the learning system passively observes a temporal sequence of input states that eventually leads to a final reinforcement or reward signal (usually a scalar). The learning system's task in this case is to predict expected reward given an observation of an input state or sequence of input states. The system may also be set up so that it can generate control signals that influence the sequence of states. In this case the learning task is usually to generate the optimal control signals that will lead to maximum reinforcement.

Delayed reinforcement learning is difficult for two reasons. First, there is no explicit teacher signal that indicates the correct output at each time step. Second, the temporal delay of the reward signal implies that the learning system must solve a temporal credit assignment problem, i.e., must apportion credit and blame to each of the states and actions that resulted in the final outcome of the sequence.

Despite these difficulties, delayed reinforcement learning has attracted considerable interest for many years in the machine learning community. The notion of a learning system interacting with an environment and learning to perform a task solely from the outcome of its experience in the environment is very intellectually appealing. It could also have numerous practical applications in areas such as manufacturing process control, navigation and path planning, and trading in financial markets.

One possible approach to temporal credit assignment is to base the apportionment of credit on the difference between temporally successive predictions. Algorithms using this approach have been termed "temporal difference" methods in Sutton (1988), and have been

studied for many years in a variety of contexts. Examples include Samuel's checkers program (Samuel, 1959) and Holland's bucket brigade algorithm (Holland, 1986). An incremental real-time algorithm called TD(λ) has been proposed in Sutton (1988) for adjusting the weights in a connectionist network. It has the following form:

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k \tag{1}$$

where P_t is the network's output upon observation of input pattern x_t at time t, w is the vector of weights that parameterizes the network, and $\nabla_w P_k$ is the gradient of network output with respect to weights. Equation 1 basically couples a temporal difference method for temporal credit assignment with a gradient-descent method for structural credit assignment. Many supervised learning procedures use gradient-descent methods to optimize network structures; for example, the back-propagation learning procedure (Rumelhart, et al., 1986) uses gradient-descent to optimize the weights in a feed-forward multilayer perceptron. Equation 1 provides a way to adapt such supervised learning procedures to solve temporal credit assignment problems. (An interesting open question is whether more complex supervised learning procedures, such as those that dynamically add nodes or connections during training, could be adapted to do temporal credit assignment.)

It can be shown that the case $\lambda = 1$ corresponds to an explicit supervised pairing of each input pattern x_t with the final reward signal z. Similarly, the case $\lambda = 0$ corresponds to an explicit pairing of x_t with the next prediction P_{t+1}. The parameter λ provides a smooth heuristic interpolation between these two limits.

Sutton provides a number of intuitive arguments why TD(λ) should be a more efficient learning procedure than explicit supervised pairing of input states with final reward. A rigorous proof is also given that TD(0) converges to the optimal predictions for a linear network and a linearly independent set of input patterns. This proof has recently been extended to arbitrary values of λ in Dayan (1992). However, no theoretical or empirical results are available for more complex tasks requiring multilayer networks, although a related algorithm called the Adaptive Heuristic Critic (Sutton, 1984) has been successfully applied to a relatively small-scale cart-pole balancing problem (Barto, Sutton & Anderson, 1983; Anderson, 1987).

The present paper seeks to determine whether temporal difference learning procedures such as TD(λ) can be applied to complex, real-world problem domains. The paper approaches this question from two perspectives. First, section 2 identifies a number of important practical issues and discusses them in the context of the current theoretical understanding of TD(λ). Some of the issues are familiar to connectionist researchers who have been studying real-world applications of supervised learning procedures. Other issues are novel and more complex than those that arise in supervised learning. Next, section 3 examines these issues in the context of a specific application: learning backgammon strategy from the outcome of self-play. This application was selected because of its complexity and stochastic nature, and because detailed comparisons can be made with the alternative approach of supervised learning from expert examples (Tesauro, 1990). We shall see that, despite a number of potentially serious theoretical and practical problems, the TD approach works

amazingly well. With zero built-in knowledge (apart from the rules), networks are able to learn to play a fairly strong intermediate-level game. The level of performance achieved not only exceeds conventional commercial programs, but perhaps more surprisingly, it also surpasses what can be achieved by supervised training on a massive data base of human expert examples. The final section discusses the implications of these results for more general practical applications, and suggests a number of directions for further research.

2. Practical issues in TD learning

2.1. Task-dependent considerations

Learning to predict and control simultaneously: A number of important practical issues in TD learning have to do with the exact nature of the task to be learned. For example, is the task purely a prediction task, or is it a combined prediction-control task? The latter is more representative of typical real-world problems, but is also presumably more difficult. The issue of simultaneously learning to predict and learning to control was not addressed in Sutton (1988), and may lie outside the scope of the TD(λ) algorithm. It might be necessary to train a separate controller network with a different algorithm while the predictor network is learning with the TD algorithm. Alternatively, one can imagine some tasks in which the output of the predictor could be used to select the control action, for example, by choosing the state with maximum expected reward. In either case it is not known whether the combined learning system would converge at all, and if so, whether it would converge to the optimal predictor/controller. It might be possible for the system to get stuck in a self-consistent but non-optimal predictor/controller.

Stationary vs. changing tasks: Another important issue is whether the task is stationary or changes over time. Also, even for fixed tasks, it is possible that the distribution of input patterns might change over time. In either case one would want the network to continually adapt in real time to changing conditions. However, there are presumably trade-offs between accuracy of learning and ability to respond to changing conditions. Such trade-offs have been extensively analyzed for the Widrow-Hoff LMS rule (Widrow, 1976), but have not been analyzed for TD learning.

Markovian vs. non-Markovian tasks: Another important issue is whether the transitions from state to state are Markovian, i.e., depend only on the current state, or whether they also depend on the history of previous states. The analysis in Sutton (1988) is only for Markovian processes, as it is pointed out that any non-Markovian process can be included within this framework by recoding the current state information so that it also contains all relevant information from previous states. However, in practice, this may make the input space so large and complex that the learning algorithm wouldn't work very well.

Multiple outcomes: The simplest types of reinforcement tasks are characterized by a binary reward signal (e.g., a success/failure signal), but more general and more complex tasks may have many different possible outcomes. The way in which these outcomes are represented in the network may be just as important as the way in which the inputs are represented. Moreover, some of the outcomes may have a much lower likelihood of occurring than other outcomes, and one might expect that such rarely occurring outcomes would

be harder to learn. In this case, one might need special techniques analogous to those used in pattern classification tasks when some of the classes have a much lower probability of occurring than other classes.

Noisy environment: A final issue is whether the environment is noisy or deterministic. Noise may appear, for example, in the rules which govern transitions from state to state, and in the final reward signal given at the terminal states. An important consideration which we examine in more detail below is the *volatility* of the stochastic environment, i.e., the step-to-step variance in expected reward. We shall see that learning is more difficult in highly volatile environments, and that a natural way to approach learning in such environments is with a look-ahead process akin to search or planning. Noise may also enter into the representation of input patterns seen by the network. This was not addressed by Sutton, and it is not known to what extent such noise degrades the learning performance.

2.2. Algorithmic considerations

Parameter tuning: As in other connectionist learning procedures, the TD(λ) algorithm has a number of adjustable parameters that have to be heuristically tuned for a given network and task. The main parameters are the learning rate α, and of course, λ itself. Ideally, one might want not just a fixed constant value of each parameter, but a schedule for varying the parameter value as a function of learning time. For example, when training a network on a stationary task, one probably needs a schedule for reducing the learning rate analogous to the $1/t$ schedules known in the stochastic approximation literature (Robbins & Monro, 1951). Also, a schedule for λ may be useful. Setting λ to a large value early in learning might help the network get off the ground quickly, while later in learning, when the predictions become more accurate, it might be better to use smaller values of λ.

Convergence: As stated previously, convergence of TD(λ) has only been proved for linear networks and linearly independent sets of input patterns. In the more general case, the algorithm may not converge even to a locally optimal solution, let alone to a globally optimal solution.

Scaling issues: Scaling considerations are often of critical importance in successful practical applications of network learning procedures. No results are available to indicate how the speed and quality of TD learning will scale with the temporal length of sequences to be learned, the dimensionality of the input space, or the dimensionality of the network (as measured, for example, by the number of weights or by the VC dimension (Vapnik & Chervonenkis, 1971)). Intuitively it seems likely that the required training time might increase very dramatically, possibly exponentially, with the sequence length. The training time might also scale poorly with the network or input space dimension, e.g., due to increased sensitivity to noise in the teacher signal. (In contrast, with perfect teacher information, we might expect the required number of training sequences to scale roughly linearly with the network's VC dimension (Blumer, et al., 1989)). Another potential problem is that the quality of solution found by gradient-descent learning relative to the globally optimal solution might get progressively worse with increasing network size.

Overtraining and overfitting: One potential advantage of the TD approach is that, unlike most applications of supervised learning, a fixed data set is not used. Instead, training takes

place on-line using patterns generated de novo. One might hope that in this situation, performance would always improve monotonically with increasing training time, i.e., overtraining would not occur. One might also hope that one could always improve the performance of the TD nets by adding more and more hidden units to the network, i.e., overfitting would not occur.

Both overtraining and overfitting may occur, however, if the error function minimized during training does not correspond to the performance function that the user cares about. For example, the performance that one cares about for a game-playing network is not how accurately it estimates the probability of winning in a given position, but rather its ability to select good moves. It may be the case that the network could produce fairly accurate predictions but not select very good moves. One would especially expect this to be true for games in which the best move is only slightly better than other alternatives. On the other hand, if the network has large errors in the absolute accuracy of its predictions, it could still be able to select good moves. This is because, as discussed in Christensen and Korf (1986), and Utgoff and Clouse (1991), a heuristic evaluation function need not exactly represent the true values of states for correct move selection. Instead, it only needs to have the correct sign for the slope between pairs of points in order to make correct state preferences. There may be many such functions, with widely varying degrees of prediction accuracy. A simple example illustrating this is shown in table 1. Overtraining and overfitting might also occur if the distribution of input training and testing patterns are different. For example, the game-playing network might be trained on its own play, but have to perform against other players. The differing styles of play would lead to different distributions of input patterns.

Incremental learning: A nice feature of TD(λ) is that the weights can be updated in a fully incremental fashion. It is not necessary to wait until the end of the sequence to compute the weight change at a given time step. However, this may not be strictly necessary in practical implementations. Modern workstations have enough memory to store the entire sequence of input and output patterns, even for fairly large problems, as long as the sequences terminate in a reasonable amount of time. If the sequence is so long that it cannot be stored, the algorithm may not be able to learn the task in any case. Thus one could imagine other algorithms that give improved performance at the price of sacrificing the fully incremental nature of TD(λ).

Table 1. An example showing the difference between a good predictor and a good move selector.

Move	True Prob.	Network #1	Network #2
A	0.846	0.842	0.719
B	0.840	0.843	0.621

The network must choose either move A or move B. The true probabilities of winning are 0.846 and 0.840 respectively. Network #1 makes highly accurate estimates of the probabilities of winning, but selects the wrong move. Network #2's estimates have large errors, but it picks the right move.

2.3. *Representational issues*

The way in which the input and output data are represented in multilayer connectionist networks has been found to be one of the most important factors in successful practical applications of supervised learning procedures. Such representational issues are likely to be equally important in practical applications of temporal difference learning. It is useful to distinguish between two basic kinds of representations: (a) *lookup table* representations, in which the network has enough adjustable parameters to explicitly store the correct output for every possible state in the input state space; and (b) *compact* representations, in which the number of adjustable parameters is much less than the number of states in the state space, and the network therefore has to capture the underlying regularity of the task.

In Sutton (1988), the TD(λ) algorithm is discussed only in terms of lookup table representations. However, the way in which TD learning works for lookup table representations is likely to be completely different from the way it works for compact representations. With lookup table representations, it is clear that the network has no way to estimate the predicted outcome of a particular state unless it has actually observed the state. Thus in order for the network to learn the entire function, it has to observe every possible state in the state space. In fact, Sutton's convergence theorem requires every possible state to be visited infinitely many times in order to guarantee convergence. This will clearly be intractable for real-world problems. Even if the state space is discrete, the number of possible states is likely to be so large that there is neither sufficient storage capacity to store the lookup table, not sufficient time to visit all possible states.

On the other hand, with compact representations, it might be possible to learn complex tasks in high-dimensional spaces reasonably well. After observing an infinitesimal fraction of all possible states, the network might be able to find a solution that generalizes acceptably for states not seen during training. Thus we can see that the ability of compact networks to generalize provides an ability to tackle otherwise intractable problems. However, there are also a number of limitations to the generalization capability of compact networks. For example, if the task is complex, a network with a limited number of hidden units might not have enough structural complexity to exactly represent the task. Also, the gradient-descent method of assigning structural credit within the network can only find local optima, not global optima. Such factors will limit the effectiveness of TD learning in complex domains.

2.4. *Volatility limit*

Let us examine equation 1 in more detail. Note that for any given input state x_t, there is a true expected outcome θ_t associated with that state, and P_t is the network's estimate of θ_t. If we had access to the values of θ_t we could use them to do back-propagation learning on the input-output pairs (x_t, θ_t). But in TD learning θ_t is not available. Instead, note that the next prediction P_{t+1} is used in a role analogous to θ_t in back-propagation. The network output P_t is being driven toward P_{t+1}, and no learning takes place when $P_t = P_{t+1}$. It should be intuitively clear that in TD learning, P_{t+1} is being used as a heuristic stochastic estimator of the true expected outcome θ_t. It should also be clear that the learning algorithm will only make progress when P_{t+1} is a more accurate stochastic estimator of θ_t than P_t is.

There are a number of reasons why P_{t+1} might not be a more accurate stochastic estimator of θ_t than P_t is. One possibility comes from the fact that the network has to generalize in any practical problem, as discussed previously. The network's generalizations for states near the end of the sequence may well be less accurate than for states far from the end.

Another reason why P_{t+1} may fail to be a more accurate estimator of θ_t is due to volatility in the environment. As learning progresses, P_t approaches θ_t, while P_{t+1} approaches θ_{t+1}. Now in highly volatile environments, θ_{t+1} may vary significantly from θ_t. It is true that the average value over all possible states that can evolve from x_t is given by $\langle \theta_{t+1} \rangle = \theta_t$, but we must remember that the network does not get to see infinitely many evolutions from x_t. In a typical complex problem, the network will only see x_t once, and thus only one state following x_t. If the value of x_{t+1} has little to do with the value of x_t due to high volatility, then P_{t+1} may be a poor estimator of θ_t. Furthermore in some problems it could be the case that volatility increases with time, so that states near the end of the sequence are more volatile than earlier states. This would provide a further limitation on the ability of P_{t+1} to estimate θ_t more accurately than P_t.

A natural way to try to overcome limitations due to volatility of the environment is to allow multiple evolutions from a given state, i.e., for each observed x_t, reset the state to x_t many times and let the stochastic process generate a distribution of successor states x_{t+1} with average expected reward $\langle P_{t+1} \rangle$. The error signal at each time step would then be proportional to $\langle P_{t+1} \rangle - P_t$. The multiple samples would provide an estimate of $\langle \theta_{t+1} \rangle$, which, as stated previously, should equal θ_t. If the temporal evolution at each step is governed by a stochastic process with a small number of possible outcomes (such as flipping a coin or rolling dice), one could explicitly compute an average over all possible outcomes.

3. A case study: TD learning of backgammon strategy

We have seen that current theory provides little indication of how TD(λ) will work in practice. In the absence of theoretical guidance, we now empirically examine the previously discussed practical issues within the context of a specific application: learning to play the game of backgammon from the outcome of self-play. Complex board games such as checkers, chess, Othello and backgammon have been widely studied as testing grounds for various machine learning procedures (Samuel, 1959; Samuel, 1967; Griffith, 1974; Quinlan, 1983; Mitchell, 1984; Frey, 1986; Christensen & Korf, 1986; Lee & Mahajan, 1988; Tesauro & Sejnowski, 1989; Tesauro, 1990). Several of these studies have employed temporal difference learning methods.

Unlike checkers, chess, and Othello, backgammon is a nondeterministic game in which the players take turns rolling dice and moving their pieces around the board as allowed by the dice roll. The first player to move all of his pieces around and off the board wins the game. The game is complicated because it is possible to "hit" opponent pieces and send them to the far end of the board, and to form blocking configurations that impede the forward movement of opponent pieces. These facts lead to a number of subtle and complex strategies and tactics at the highest levels of play (Magriel, 1976).

Backgammon offers a number of attractive features as a test vehicle for TD learning approaches. Due to the stochastic dice rolls, the evolution of board states during the course

of a backgammon game can be characterized as an absorbing Markov process, in which the initial state is always a unique starting configuration, and in which the subsequent transitions proceed randomly, depending only on current state information, until a well-defined terminal state is reached, characterized by one side having all its pieces off the board. This is precisely the type of situation for which TD(λ) was designed. Playing the game at expert level involves considerable complexity, but it has been demonstrated (Berliner, 1979; Tesauro & Sejnowski, 1989) that much of this complexity can be captured in a static evaluation function and does not require deep look-ahead searches. This means that a feed-forward neural network could learn a static evaluation function that would play well without search, and that the quality of learning can be directly assessed by measuring game performance. (In contrast, the performance of a search-based program depends on both the quality of the evaluation function and the power of the search procedure.) Finally, it is possible to make a detailed comparison of TD learning with the alternative approach of supervised learning from expert examples (Tesauro, 1990). This is important for general practical applications, because in order for TD learning to be successful in the real world, it not only has to work well on hard problems, but it also has to be competitive with other approaches such as supervised learning.

It seems reasonable that, by watching two fixed opponents play out a large number of games against each other, a network could learn by TD methods to predict the expected outcome of any given board position. In addition to estimating expected outcome, such a network could also be used for move selection by generating all legal moves in a given position and picking the move with the maximum expected outcome. A more interesting learning question, however, is whether the network could learn from the outcome of its own play. As the network learns, its control strategy changes, and thus the distribution of input patterns and final rewards would also change. This is the type of learning that will be examined in this section, even though it is not clear a priori that such a learning system would converge to a sensible solution.

3.1. Set-up of the learning system

The TD(λ) algorithm can be applied to backgammon in the following straightforward way: a network is set up to observe a sequence of board positions $x_1, x_2, \ldots, x_f$, resulting in a final reward signal z. In the simplest case the reward signal is $z = 1$ if White wins and $z = 0$ if Black wins. In this case the network's output P_t is an estimate of White's probability of winning from board position x_t. The sequence of board positions is generated by setting up an initial configuration, and making plays for both sides using the network's output as an evaluation function. In other words, the move selected at each time step is the move that maximizes P_t when White is to play and minimizes P_t when Black is to play.

A critical factor in the overall performance of the learning system is the representation scheme used to encode the input board description. It is well known in computer games research that significantly higher levels of performance can be achieved if the board state is described using "features" relevant to good play, as opposed to a "raw" board description. In the experiments reported here, however, the input encoding schemes only contained simple encodings of the raw board information (the number of White or Black men at each location) and did not utilize any additional pre-computed features.

Since the input encoding scheme contains no built-in knowledge about useful features, and since the network only observes its own play, we may say that this is a "knowledge-free" approach to learning backgammon. Such an approach is interesting because it is not clear that it can make any progress at all beyond its starting state, which for a network with random initial weights is a random move selector. The zero-knowledge approach also provides an important baseline for judging other approaches using various forms of built-in knowledge.

The approach described above is similar in spirit to Samuel's approach to learning checkers from self-play, but in several ways it is a more challenging learning task. One important difference is that Samuel's board description was in terms of a number of hand-crafted features, several of which were designed in consultations with human checkers experts. However, the networks studied here use only a raw board description and had no knowledge built into the input features. The evaluation function learned in Samuel's study was a linear function of the input variables, whereas multilayer networks learn more complex nonlinear functions. Also, the final reward signal in backgammon is noisy due to the dice rolls; this presumably makes the learning task more difficult than in noise-free games such as checkers. The branching ratios in backgammon are so large that look-ahead methods cannot be employed, whereas Samuel used search both in move selection and in calculation of the learning algorithm's error signal. Finally, Samuel found that it was necessary to give the learning system at least one fixed intermediate goal, material advantage, as well as the ultimate goal of the game. The proposed backgammon learning system has no such intermediate goals.

3.2. Learning disengaged bearoff strategy

Like many other games, the full complexity of backgammon is greatly simplified in certain special situations. For example, finding good moves in racing situations, in which hitting and blocking are not possible, is considerably easier than in fully engaged positions, in which hitting and blocking are possible. The first TD experiments we shall examine are designed to learn the case of disengaged bearoff positions, in which both sides have all of their men in the their home quadrant and can remove them from the board. This is an exceedingly simple situation because there is a simple yet strong heuristic for correct play: always select the move that takes the maximum number of pieces off the board. This principle is only violated in certain rare situations. If the principle does not uniquely determine a move, a secondary consideration is to distribute the remaining men as smoothly as possible; this usually determines the best move. (An interesting exception to this rule is discussed in Berliner (1977).)

Another advantage of studying this situation is that it can be solved essentially exactly by conventional algorithms (Berliner, 1977; Zadeh, 1977). For each of the approximately 54,000 possible bearoff configurations for a given side, one can recursively compute and store in a table the exact probability of removing all men on one roll, two rolls, three rolls, etc.. One can then use this information to compute the exact probability that either side will win. Thus, in addition to comparisons with supervised learning of expert examples, one can also compare the results of TD learning with the exact optimal move choices and probabilities of winning.

The networks trained on this task had 16 units to encode the board description: 6 units to encode the number of Black men at locations 1–6, 6 units to encode the number of White men at locations 19–24, 2 units to encode the number of White and Black men off the board, and 2 units to encode the side to move. The networks had a feed-forward structure with full connectivity from one layer to the next. Two architectures were examined: a single-layer architecture with direct input-output connections and no hidden units, and a multi-layer architecture containing a single hidden layer with varying numbers of hidden units. The single-layer architecture can only represent linearly separable functions (Minsky & Papert,1969), while the multilayer architecture, given sufficient hidden units, is capable of universal function approximation (Hornik, Stinchcombe & White, 1989). Both the hidden units and the output unit utilized a sigmoidal transfer function $y = 1/(1 + e^{-x})$.

The initial board configurations were generated randomly by distributing all 15 men for each side with uniform probability in the six home board locations. With this distribution of initial positions, the average sequence length is about 14 time steps with good play on both sides.

As with back-propagation, a certain amount of parameter tuning was required to get good results with the TD(λ) algorithm. It was found that a learning rate of $\alpha = 0.1$ usually gave good results. Lower learning rates did not improve the maximum performance (although they did reduce the level of stochastic fluctuations in performance), whereas significantly higher learning rates did degrade performance. Also, the value of λ appeared to have almost no effect on the maximum obtainable performance, although there was a speed advantage to using large values of λ. The results reported here used a value of λ set (somewhat arbitrarily) at 0.7. The initial random weight scale also did not appear to be important; in the results reported here, weights were initialized to uniform random values in the range $[-0.5, +0.5]$.

Two measures of performance were monitored to assess the results of learning. The absolute prediction error was monitored by comparing the network's outputs for the positions seen during training with the exact win probabilities computed from the lookup tables. (One should keep in mind that these exact probabilities were not used as part of the training signal.) The network's move selection ability was also monitored during training by measuring the fraction of time the network selected the theoretically optimal move in a fixed set of test positions. (There were 210 test positions, taken from a set of games in which the author played both sides. This test set was pruned from a larger set of over 300 positions by deleting all positions with no unique best move, according to the lookup tables. In most of the deleted positions, the probability of winning was either exactly 1 or exactly 0 regardless of which move was selected.)

A perhaps surprising discovery was that, apart from stochastic fluctuations controlled by the value of α, the absolute prediction error always decreased during training. On the other hand, the move-selection performance usually reached a maximum after several tens of thousands of games, and then decreased thereafter. (Due to the stochastic training procedure and the relatively flat nature of the learning curves, it is difficult to say precisely when the maximum performance was reached.) As stated previously, this "overtraining" could be due to the differing distributions of training and test positions, or it could also be due to the difference between the function minimized by the learning algorithm (prediction error) and the function that the user cares about (move-selection ability).

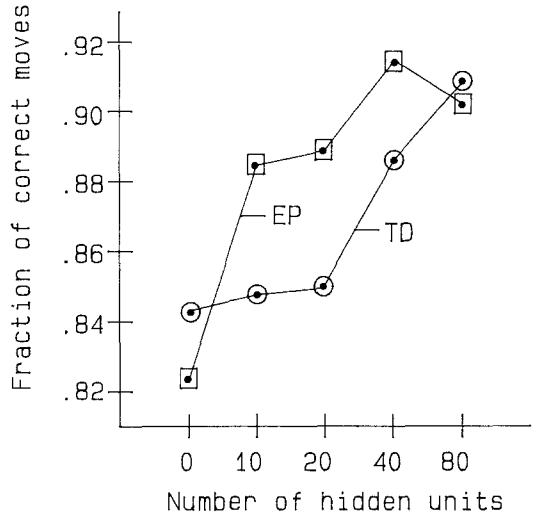


Figure 1. Plot of move selection performance vs. number of hidden units for networks trained on disengaged bearoff positions using TD learning from self-play (TD), and supervised training on human expert preferences (EP). Both learning systems used identical 16-unit coding schemes.

The level of move-selection performance obtained by the networks was also surprising in view of the average absolute prediction errors. Typically the networks are only able to estimate the probability of winning to within about 10% of the true probability. On the other hand, one usually needs to make discriminations at the 1% level or lower for accurate move selection. Thus based on this consideration alone, one would expect the networks to be rather poor move selectors. However, figure 1, which plots move-selection performance as a function of the number of hidden units for TD-trained networks, shows that performance is in fact quite good. Furthermore, the networks with hidden units clearly do better than networks without hidden units. This indicates that the networks have absorbed some of the nonlinear aspects of the problem, and are not just implementing a linear rule such as "maximize the number of pieces taken off the board." It is also reassuring that this nonlinear structure can be learned in the TD procedure, despite the lack of any theoretical guarantees.

Also plotted for comparison are the results of supervised training of identical networks with identical coding schemes on a data set of about 1700 training positions. In each training position, a human expert (the author) recorded a preference of best move. The training methodology is the "comparison paradigm" described in Tesauro (1989), in which the network is trained by back-propagation to score the expert's move higher than each of the other legal moves.

We can see that, as is usual with back-prop nets trained on a fixed data set, the ability to increase performance by adding more hidden units is eventually lost for sufficiently large nets: at this point, the networks start to overfit the training data. On the other hand, since the TD nets are not trained on a fixed data set, the performance can in principle always be improved by adding more hidden units. We do in fact find that, at least for the network

sizes studied here, performance increases monotonically with the number of hidden units, and that the TD nets eventually catch up to the performance level of the nets trained on expert preferences (denoted EP in figure 1).

One should note that the absolute prediction error of the TD nets generally gets worse as the end of the game approaches. This is not surprising, because states far from the end can be accurately represented by a simple pip count approximation, but this breaks down when there are only a few pieces left. Also for this particular task, states near the end of the game have higher volatility than states far from the end. As stated previously, these factors may provide important limits to the ability of TD learning to approach theoretically optimal performance.

3.3. Learning full-board racing strategy

The results of TD training of disengaged bearoff situations were encouraging enough to attempt the next logical extension: training of general racing situations covering the entire board. These situations require many more units to encode the board description, and they are usually more complicated because of the possibility of bonus wins called "gammons." These occur when one side removes all its pieces before the other side takes off any pieces, and the point value awarded is double the normal value. (There is also the possibility of winning a triple-value "backgammon," but this will not be considered here.) Thus in a general racing situation, if one has men in the outer quadrants, one has to choose between playing for a win and avoiding the loss of a gammon. Strategies in these two cases are usually different. For example, in gammon avoidance situations it is usually correct to bear men into the 6 point, whereas if there is no gammon risk, one should distribute one's men evenly on the 6, 5 and 4 points.

The possibility of winning a gammon means that a game can now end in one of four possible outcomes. A straightforward way to handle this is to use a four-component reward signal, with each component corresponding to one of the four possible outcomes. One would then train a network with four output units, which would learn to estimate for any input position the probabilities of the four separate outcomes. (Without additional constraints, however, there would be no guarantee that the network's estimated probabilities would sum to 1.) For move-making decisions, one could use the network's estimated probabilities to estimate expected payoff, and select the move that maximizes expected payoff. Specifically, if (p_1, p_2, p_3, p_4) are the network's estimates of the probabilities of (W wins, W gammons, B wins, B gammons), then the expected payoff to White is given by $p_1 + 2p_2 - p_3 - 2p_4$. Thus even in this more complex situation, one can still cast the problem as a prediction learning problem only, and can use the predictor network's outputs to generate control decisions without training a separate control network.

For TD training of general race situations, a 52-unit coding scheme was used: 24 units each to encode the number of White or Black men at locations 1–24, 2 units to encode White and Black men off, and 2 units to encode the player to move. A modified four-component reward signal was used in which the units representing regular wins were activated if a side had either a regular or a gammon win. This should be easier to learn for a network with sigmoidal units, since a unit that only represents regular wins has to learn a

non-monotonic function of the lead in the race. The initial board configurations were generated by randomly picking a divider location on the board, and randomly distributing all White men uniformly to one side and all Black men uniformly to the other side of the divider location. With this distribution of starting positions, the average number of time steps from start to finish was about 20, as compared to 14 in the bearoff experiments. As in the previous section, the algorithm parameters were set at $\alpha = 0.1$ and $\lambda = 0.7$.

In this more complex situation, algorithms are no longer available for computing the exact outcome probabilities. Hence the only performance measure used was the fraction of time the network agreed with a human expert (the author) on the best move in a fixed set of 248 full-board racing test positions. This test set measure is less reliable than in the previous section, because there may be significant human errors in the choice of best move. There may also be a substantial fraction of positions in which the outcome is uniquely determined regardless of which move is made.

Results of TD training are shown in figure 2, which plots maximum test set performance as a function of number of hidden units. Once again these results are compared with networks using an identical coding scheme trained on a data set of about 3100 human expert examples. Once again the TD networks reached peak performance after several tens of thousands of games. For this task, no clear evidence of overtraining was seen; thus with further training it might be possible to obtain higher levels of performance.

We can see that, despite the increased size of the input space, complexity of the task, and length of the sequence, the TD networks still were able to achieve a high level of performance. The test set performance is almost as good as the EP nets. In fact, since the test set measure is admittedly biased in favor of the expert-trained nets (which could pick

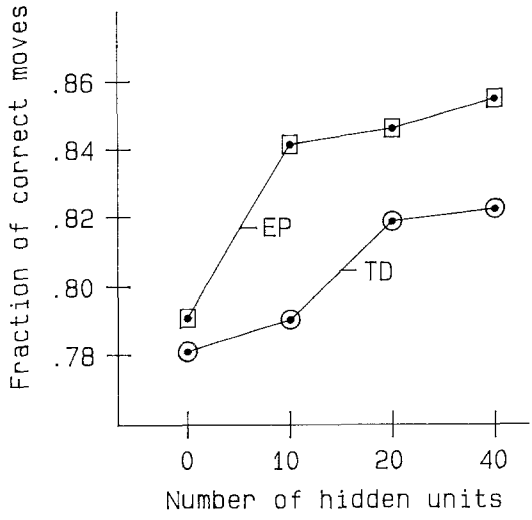


Figure 2. Plot of move selection performance vs. number of hidden units for networks trained on full-board racing positions using TD learning from self-play (TD), and supervised training on human expert preferences (EP). Both learning systems used identical 52-unit coding schemes.

up consistently wrong or arbitrary stylistic preferences of the expert), one may legitimately wonder whether the TD net really is inferior in an absolute sense, and if so, by how much. Also, further improvements in TD performance might be obtainable by adding more hidden units to the TD networks, by training longer, or by using a more representative distribution of starting positions.

Qualitatively, the TD nets appear to have discovered many important ingredients of general racing strategies. The main apparent defect is a tendency to waste pips when the network has both a small chance of winning and a small chance of losing a gammon. In this case a human would try to remove all doubt about saving the gammon, whereas the network tends to make plays that suggest trying to win.

3.4. Learning to play the entire game

We now consider extending TD learning to cover more general engaged situations, in which hitting and blocking are possible. These situations are much more complex than racing situations, and the sequences can be much longer. (In typical human play the average sequence length from start to finish is around 50–60 time steps, whereas for random move selectors, games can last several hundred or even thousands of time steps.) Also the optimal strategy for one side depends more critically on what strategy the opponent is using, whereas in racing situations the choice of best move is largely independent of the opponent's strategy. This means that a network adapting to its own play has a more serious possibility of getting stuck in self-consistent but non-optimal strategies.

The networks trained on this task used an expanded scheme to encode the local information. Rather than a single unit to encode the number of men of a given color at a given location, a truncated unary encoding with four units was used. The first three units encoded separately the cases of one man, two men, and three men, while the fourth unit encoded the number of men beyond 3. (In the development of Neurogammon, it was found that truncating at 5 or 6 units rather than 4 units gives better performance but of course takes longer to simulate.) This coding scheme thus used 96 units for each side to encode the information at locations 1–24, and an additional 6 units to encode the number of men on the bar, off the board, and the player to move, for a total of 198 input units.

The parameter settings in these experiments were once again $\alpha = 0.1$ and $\lambda = 0.7$. A few experiments with smaller and larger values of λ seemed to indicate that larger values would decrease the performance, while smaller values would give about the same performance.

Networks were trained on the entire game, starting from the opening position and going all the way to the end. This is an admittedly naive approach, and given the complexity of this task, one might wonder whether it would actually work. Alternatively, one can consider dividing the game into a number of phases (e.g., early engaged, middle engaged, late engaged, and race), and using as a heuristic reward for a given phase the network output for the next phase. Some preliminary experiments were performed with a two-phase approach in which the network was trained up to the point of disengagement, and the output of a previously trained racing net (described in the previous section) was used as a heuristic reward. In this way the network would see a shorter sequence of positions, and its evaluation function would be simpler than the evaluation function needed for the entire

game. On the other hand, there may be problems due to the fact that the racing network was trained with a somewhat arbitrary distribution of starting positions. Furthermore, any systematic biases in the racing network's judgement might be transferred to the engaged network. Empirically this two-phase approach seemed to offer some potential for improving the performance of smaller nets, but not for larger nets, and view of the above-mentioned theoretical concerns, this approach was eventually abandoned.

Training a single net on the entire game was not expected to yield any useful results other than a reference point for judging more sensible approaches. However, the rather surprising result was that a significant amount of learning actually took place. Performance on the 248-position racing test set reached about 65%. (This is substantially worse than the racing specialists described in the previous section.) Performance on a separate set of 500 full-contact test positions is plotted in figure 3. Again these figures are compared with results of supervised training on a human expert training set containing over 15,000 engaged positions. We can see that the TD nets reached levels of performance well beyond the initial random networks, showing that substantial learning took place. (The random initial networks only select the right move about 5% of the time.) The performance levels lag somewhat behind what can be achieved with expert preference training, but we must remember that this test set measure may not give the most accurate assessment of true game-playing strength.

A more accurate and objective measure of game-playing strength is actual game performance against an opponent. Both the TD nets and the EP nets have been tested in actual game play against Sun Microsystem's Gammontool program. Gammontool is representative of the level of performance that is typically achieved with conventional commercial programs,

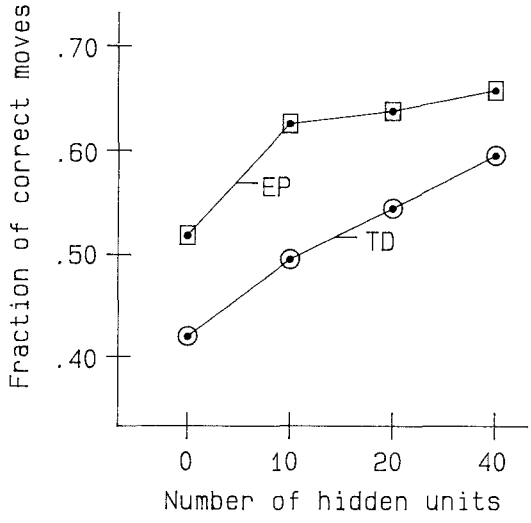


Figure 3. Plot of move selection performance in general engaged positions vs. number of hidden units for networks trained using TD learning from self-play (TD), and supervised training on human expert preferences (EP). Both learning systems used identical 198-unit coding schemes.

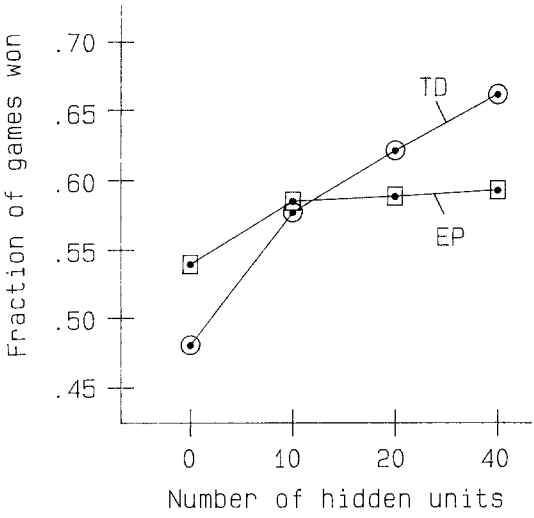


Figure 4. Plot of game performance against Gammontool vs. number of hidden units for networks trained using TD learning from self-play (TD), and supervised training on human expert preferences (EP). Each data point represents the result of a 10,000 game test, and should be accurate to within one percentage point.

and provides a decent benchmark for measuring game-playing strength. Human beginners can win about 40% of the time against it, decent intermediate-level humans would win about 60%, and human experts would win about 75%. (The random initial networks before training win only about 1%.) Since the EP nets are trained on engaged positions only, the testing procedure is to play out the game with the network until it becomes a race, and then use Gammontool's algorithm to move for both sides until the end. This also does not penalize the TD net for having learned rather poorly the racing phase of the game. Results are plotted in figure 4.

Given the complexity of the task, size of input space and length of typical sequences, it seems remarkable that the TD nets can learn on their own to play at a level substantially better than Gammontool. Perhaps even more remarkable is that the TD nets surpass the EP nets trained on a massive human expert data base: the best TD net won 66.2% against Gammontool, whereas the best EP net could only manage 59.4%. This was confirmed in a head-to-head test in which the best TD net played 10,000 games against the best EP net. The result was 55% to 45% in favor of the TD net. This confirms that the Gammontool benchmark gives a reasonably accurate measure of relative game-playing strength, and that the TD net really is better than the EP net. In fact, the TD net with no features appears to be as good as Neurogammon 1.0, backgammon champion of the 1989 Computer Olympiad, which does have features, and which wins 65% against Gammontool. A 10,000 game test of the best TD net against Neurogammon 1.0 yielded statistical equality: 50% for the TD net and 50% for Neurogammon.

The TD net's performance against these three opponents indicates that it has reached a significant level of playing ability. This violates a widely held belief in computer games and machine learning research that significant levels of performance in game-playing programs

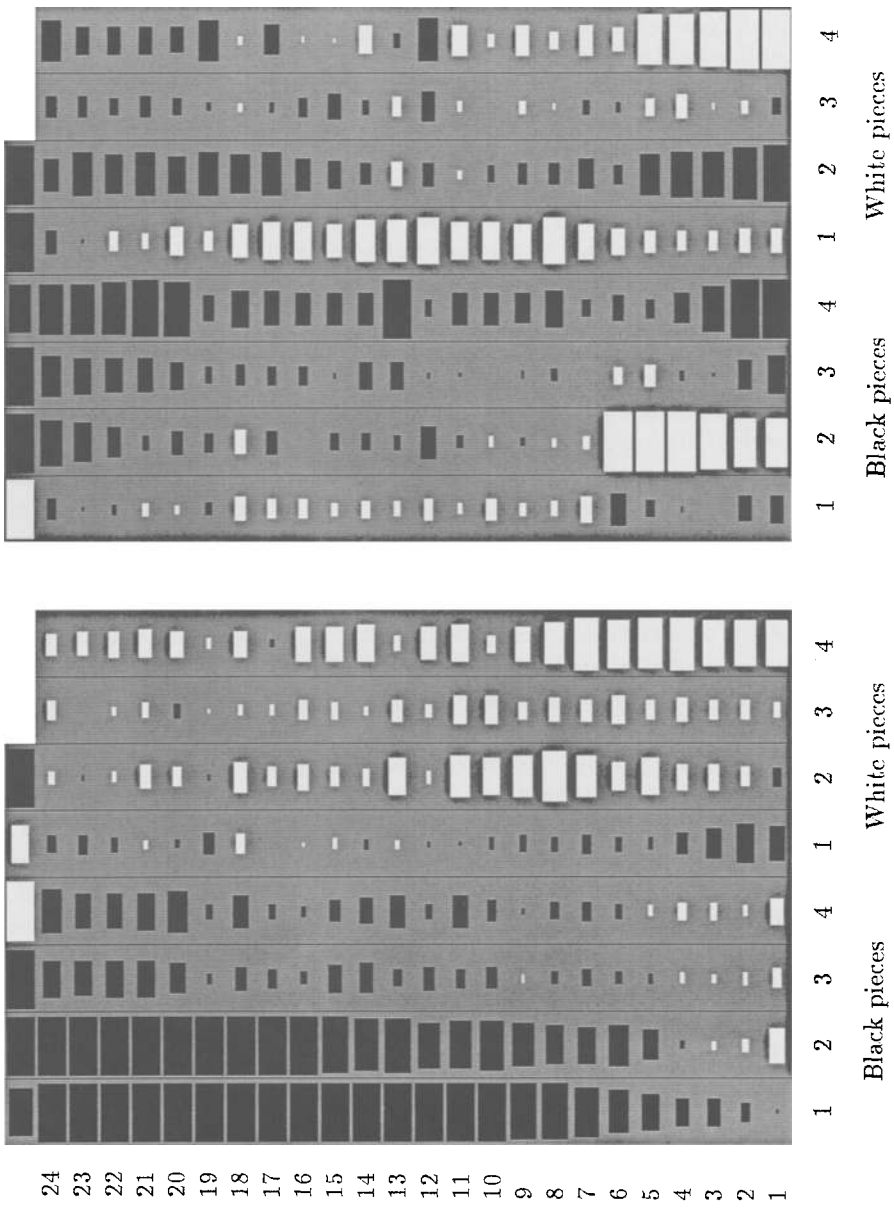
can only be achieved through the use of hand-crafted features in the evaluation function. Apparently the hidden units in the TD net have discovered useful features through self-play. When one looks at the pattern of weights learned by the TD net, one can see a great deal of spatially organized structure, and some of this structure can be interpreted as useful features by a knowledgeable backgammon player. Figure 5 shows the weights from the input layer to two of the hidden units in the best TD net. Both hidden units contribute positively to the estimation of Black's chances of winning and gammoning, and negatively to the estimation of White's chances of winning and gammoning. The first hidden unit appears to be a race-oriented feature detector, while the second hidden unit appears to be an attack-oriented feature detector.

The training times needed to reach the levels of performance shown in figure 4 were on the order of 50,000 training games for the networks with 0 and 10 hidden units, 100,000 games for the 20-hidden unit net, and 200,000 games for the 40-hidden unit net. Since the number of training games appears to scale roughly linearly with the number of weights in the network, and the CPU simulation time per game on a serial computer also scales linearly with the number of weights, the total CPU time thus scales quadratically with the number of weights: on an IBM RS/6000 workstation, the smallest network was trained in several hours, while the largest net required two weeks of simulation time.

The networks did not appear to overtrain, but it is not clear whether the performance increases very slowly with further training or stays flat. Since the networks in the previous sections also required several tens of thousands of training games, this suggests that the number of training sequences needed to train a TD network may not scale badly with the task complexity or average sequence length. Instead, the training time may just depend on the number of bits of information needed to specify a trained network, and on how many bits of information are received per game. Since the outcome of each game gives one bit of information (two bits including gammons), and since the networks have several thousand weights that probably must be specified to at least 4–8 bits of accuracy, this suggests a training time on the order of tens of thousands of games.

Some qualitative observations on the styles of play learned by the TD and EP nets are worth noting. The TD nets have developed a style emphasizing running and tactical play. For example, it prefers to immediately split its back men rather than bringing down builders or slotting home board points. It is good in running game situations and in tactical situations such as blot-hitting contests and blitz attacks. The EP nets, however, favor a more quiescent positional style of play emphasizing blocking rather than racing. This is more in line with human expert play (at least the particular human expert who made the training data), but it often leads to complex prime vs. prime and back-game situations that are hard for the network to evaluate properly. This suggests one possible advantage of the TD approach over the EP approach: by learning to imitate an expert teacher, the learning system may get itself into situations that it doesn't know how to handle. With the alternative approach of learning from experience, the learner may not reproduce the expert strategies, but at least it will learn to handle whatever situations are brought about by its own strategy.

It's also interesting that TD net ended up playing well in the early engaged phase of play, whereas its play in the late racing phase is rather poor. This is contrary to the intuitive notion that in temporal credit assignment learning, states far from the end of the sequence will be harder to learn than states near the end. Apparently the inductive bias due to the


24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
1 2 3 4 1 2 3 4
Black pieces
White pieces
1 2 3 4 1 2 3 4
Black pieces
White pieces

representation scheme and network architecture is more important than temporal distance to the final outcome. This may partially explain why training times were not dramatically worse in the full-game situation than in the simplified endgame situations.

4. Conclusions

We have seen that, from the theoretical point of view, there may be a number of important limitations to the effectiveness of TD learning in large-scale complex domains. The algorithm may not converge even for prediction only tasks, let alone for combined prediction/control tasks. Even if the algorithm does converge, it may get stuck in poor locally optimal solutions. Finally, even if the algorithm is able to find good solutions, the scaling of required training time with problem size or sequence length may be so poor that the learning task would be effectively intractable.

In view of these potential difficulties, there are a number of very encouraging conclusions that can be drawn from the backgammon application. Empirically the algorithm always converges to at least a local minimum. The quality of solution found by the network is usually fairly good, and generally improves with increasing numbers of hidden units. Furthermore, the scaling of training time with the length of input sequences, and with the size and complexity of the task, does not appear to be a serious problem. Finally, it is encouraging that the network was able to learn to make good control decisions as well as learn to estimate expected outcome. In fact, the network's ability to select good moves is much better than we have a right to expect, because its absolute accuracy in estimating expected outcome is usually at the 10% level, whereas the difference in expected outcome between optimal and non-optimal moves is usually at the level of 1% or less. This suggests that much of the network's absolute prediction error is systematic error that applies equally to all moves generated from the same initial position, and thus cancels out in move-making decisions.

The most encouraging finding, however, is a clear demonstration that in the full-game situation, the TD learning network with zero built-in knowledge can achieve a higher level of overall game performance than an identical network trained on a massive data base of human expert examples. The level of performance achieved is in fact equal to a program that convincingly won the backgammon championship in a major tournament for computer programs. This level of performance is so far beyond what one would have expected beforehand that it seems worthwhile to devote some further effort to understanding exactly how

Figure 5. A diagram illustrating the weights from the input units to two of the 40 hidden units in the best TD net. Black squares represent negative weights and white squares represent positive weights; the size of the square indicates the magnitude of the weights. The rows represent from bottom to top the spatial locations 1–24. The top row represents: (W barmen, B barmen, W men off, B men off, W turn, B turn). The columns represent the number of Black and White men as indicated. The first hidden unit has two noteworthy features: a linearly increasing pattern of negative weights for Black blots and Black points, and a negative weighting of White men off and a positive weighting of Black men off. These features are recognizable as contributing to an estimate of Black's probability of winning based on his lead in the race. The second hidden unit has the following noteworthy features: strong positive weights for Black home board points, strong positive weights for White men on bar, positive weights for White blots, and negative weights for White points in Black's home board. The experienced backgammon player recognizes that these factors all contribute to the probability of a successful Black attacking strategy.

this is possible. It may also be worthwhile trying to apply the combination of TD with back-propagation, or TD with other supervised learning algorithms, to other difficult temporal credit assignment problems.

Future prospects for the backgammon application look very promising. It certainly seems possible that further improvements in performance can be obtained merely by adding more hidden units to the network and training for longer training times, although the quadratic scaling of CPU time with the number of weights may limit how far this can be carried in practice. Better results might also be obtainable by optimizing the parameter settings, or by modifications of the TD training procedure. For example, the next prediction could be replaced by an average over all possible dice rolls; this could reduce limitations due to volatility. Also, partitioning the game into a number of temporal phases and training separate specialist networks on each phase may make it easier for each specialist network to learn the evaluation function appropriate for that particular phase. In actual game play, the outputs of the specialists could be combined using smoothly-varying application coefficients, as suggested in Berliner (1979). Finally, an improved representation scheme, which uses features known to be useful in backgammon evaluation functions, and which is better able to represent the value of near-end states, might give substantially better results. Such improved representation schemes are currently under investigation. As this article goes to press, a TD net containing all of Neurogammon's features is learning from self-play. The network has reached 71% against Gammontool and continues to improve slowly with further training. It appears to be the strongest program ever seen by this author.

Beyond this specific application, however, the larger and more important issue is whether learning from experience can be useful and practical for more general complex problems. Certainly the quality of results obtained in this study suggests that the approach may work well in practice, and may work better than we have a right to expect theoretically. There may also be some intrinsic advantages to this approach over supervised training on a fixed set of labeled exemplars. At the very least, for tasks in which the exemplars are hand-labeled by humans, it eliminates the laborious and time-consuming process of labeling the data. Furthermore, the learning system would not be fundamentally limited by the quantity of labeled data, or by any intrinsic errors in whatever process is used to label the data. Also, the learning system might be able to avoid getting itself into situations that it doesn't know how to handle, as can happen when one is learning by imitating an expert. Finally, preserving the intrinsic temporal nature of the task, and informing the system of the consequences of its actions, may convey important information about the task that is not necessarily contained in the labeled exemplars. More theoretical and empirical work will be needed to establish the relative advantages and disadvantages of the two approaches. A result of this may be the development of novel learning paradigms combining supervised learning with learning from outcome; in combination it might be possible to surpass what either approach could achieve individually. Preliminary work supporting this hypothesis is reported in Utgoff and Clouse (1991).

Acknowledgments

The author thanks Scott Kirkpatrick and Rich Sutton for helpful comments on an earlier version of the manuscript.

References

Anderson, C.W. (1987). Strategy learning with multilayer connectionist representations. *Proceedings of the Fourth International Workshop on Machine Learning* (pp. 103–114).

Barto, A.G., Sutton, R.S., & Anderson, C.W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man and Cybernetics, 13,* 835–846.

Berliner, H. (1977). Experiences in evaluation with BKG—a program that plays backgammon. *Proceedings of IJCAI* (pp. 428–433).

Berliner, H. (1979). On the construction of evaluation functions for large domains. *Proceedings of IJCAI* (pp. 53–55).

Blumer, A., Ehrenfeucht, A., Haussler, D., & Warmuth, M. (1989). Learnability and the Vapnik-Chervonenkis dimension. *JACM, 36,* 929–965.

Christensen, J. & Korf, R. (1986). A unified theory of heuristic evaluation functions and its application to learning. *Proceeding of AAAI-86* (pp. 148–152).

Dayan, P. (1992). The convergence of TD(λ). *Machine Learning, 8,* 341–362.

Frey, P.W. (1986). Algorithmic strategies for improving the performance of game playing programs. In: D. Farmer, et al. (Eds.), *Evolution, games and learning.* Amsterdam: North Holland.

Griffith, A.K. (1974). A comparison and evaluation of three machine learning procedures as applied to the game of checkers. *Artificial Intelligence, 5,* 137–148.

Holland, J.H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In: R.S. Michalski, J.G. Carbonell & T.M. Mitchell, (Eds.), *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos, CA: Morgan Kaufmann.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks, 2,* 359–366.

Lee, K.-F. & Majahan, S. (1988). A pattern classification approach to evaluation function learning. *Artificial Intelligence, 36,* 1–25.

Magriel, P. (1976). *Backgammon.* New York: Times Books.

Minsky, M.L. & Papert, S.A. (1969). *Perceptrons.* Cambridge, MA: MIT Press. (Republished as an expanded edition in 1988).

Mitchell, D.H. (1984). Using features to evaluate positions in experts' and novices' Othello games. Master's Thesis, Northwestern Univ., Evanston, IL.

Quinlan, J.R. (1983). Learning efficient classification procedures and their application to chess end games. In: R.S. Michalski, J.G. Carbonell & T.M. Mitchell (Eds.), *Machine learning.* Palo Alto, CA: Tioga.

Robbins, H. & Monro, S. (1951). A stochastic approximation method. *Annals of Mathematical Statistics, 22,* 400–407.

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In: D. Rumelhart & J. McClelland, (Eds.), *Parallel distributed processing.* Vol. 1. Cambridge, MA: MIT Press.

Samuel, A.(1959). Some studies in machine learning using the game of checkers. *IBM J. of Research and Development, 3,* 210–229.

Samuel, A.(1967). Some studies in machine learning using the game of checkers, II—recent progress. *IBM J. of Research and Development, II,* 601–617.

Sutton, R.S. (1984). Temporal credit assignment in reinforcement learning. Doctoral Dissertation, Dept. of Computer and Information Science, Univ. of Massachusetts, Amherst.

Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning, 3,* 9–44.

Tesauro, G. & Sejnowski, T.J. (1989). A parallel network that learns to play backgammon. *Artificial Intelligence, 39,* 357–390.

Tesauro, G. (1989). Connectionist learning of expert preferences by comparison training. In D.Touretzky (Ed.), *Advances in neural information processing, 1,* 99–106.

Tesauro, G. (1990). Neurogammon: a neural network backgammon program. *IJCNN Proceedings,* III, 33–39.

Utgoff, P.E. & Clouse, J.A. (1991). Two kinds of training information for evaluation function training. To appear in: *Proceedings of AAAI-91.*

Vapnik, V.N. & Chervonenkis (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory Prob. Appl., 16,* 264–280.

Widrow, B., et al. (1976). Stationary and nonstationary learning characteristics of the LMS adaptive filter. *Proceedings of the IEEE, 64,* 1151–1162.

Zadeh, N. & Kobliska, G. (1977). On optimal doubling in backgammon. *Management Science, 23,* 853–858.