ELSEVIER

Artificial Intelligence 134 (2002) 181–199

Artificial Intelligence

www.elsevier.com/locate/artint

Programming backgammon using self-teaching neural nets

Gerald Tesauro

IBM Thomas J. Watson Research Center, 30 Saw Mill River Rd., Hawthorne, NY 10532, USA

Abstract

TD-Gammon is a neural network that is able to teach itself to play backgammon solely by playing against itself and learning from the results. Starting from random initial play, TD-Gammon's self-teaching methodology results in a surprisingly strong program: without lookahead, its positional judgement rivals that of human experts, and when combined with shallow lookahead, it reaches a level of play that surpasses even the best human players. The success of TD-Gammon has also been replicated by several other programmers; at least two other neural net programs also appear to be capable of superhuman play.

Previous papers on TD-Gammon have focused on developing a scientific understanding of its reinforcement learning methodology. This paper views machine learning as a tool in a programmer's toolkit, and considers how it can be combined with other programming techniques to achieve and surpass world-class backgammon play. Particular emphasis is placed on programming shallow-depth search algorithms, and on TD-Gammon's doubling algorithm, which is described in print here for the first time.

Keywords: Reinforcement learning; Temporal difference learning; Neural networks; Backgammon; Games; Doubling strategy; Rollouts

1. Introduction

Complex board games such as Go, chess, checkers, Othello and backgammon have long been regarded as great test domains for studying and developing various types of machine learning procedures. One of the most interesting learning procedures that can be studied in such games is reinforcement learning from self-play. In this approach, which originated long ago with Samuel's checkers program [18], the program plays many games against itself, and uses the "reward" signal at the end of each game to gradually improve the quality of its move decisions.

E-mail address: tesauro@watson.ibm.com (G. Tesauro).

0004-3702/02/$ – see front matter
PII: S0004-3702(01)00110-2

This paper presents TD-Gammon, a self-teaching program that was directly inspired by Samuel's research. TD-Gammon is a neural network that trains itself to be an evaluation function for the game of backgammon, by playing against itself and learning from the outcome. It combines two major developments of recent years that appear to overcome traditional limitations to reinforcement learning. First, it uses the Multi-Layer Perceptron neural net architecture, widely popularized in backpropagation learning, as a method of learning complex nonlinear functions of its inputs. Second, it apportions "temporal credit assignment" during each self-play game using a "Temporal Difference" (or simply TD) learning methodology [23]. The basic idea of TD methods is to base learning on the difference between temporally successive predictions. In other words, the goal is to make the learner's current prediction for the current input pattern more closely match the subsequent prediction at the next time step. The specific TD method used, which will be described later in more detail, is the TD(λ) algorithm proposed in [22].

TD-Gammon was originally conceived as a basic-science study of how to combine reinforcement learning with nonlinear function approximation. It was also intended to provide a comparison of the TD learning approach with the alternative approach of supervised training on a corpus of expert-labeled exemplars. The latter methodology was used previously in the development of Neurogammon, a neural network backgammon program that was trained by backpropagation on a data base of recorded expert move decisions. Its input representation included both the raw board information (number of checkers at each location), as well as several hand-crafted "features" that encoded important expert concepts. Neurogammon achieved a strong intermediate level of play, which enabled it to win in convincing style the backgammon championship at the 1989 International Computer Olympiad [24]. By comparing TD-Gammon with Neurogammon, one can get a sense of the potential of TD learning relative to the more established approach of supervised learning.

Despite the rather academic research goals listed above, TD-Gammon ended up having a surprising practical impact on the world of backgammon. The self-play training paradigm enabled TD-Gammon's neural net to significantly surpass Neurogammon in playing ability. The original version 1.0 of TD-Gammon, which was trained for 300,000 self-play games, reached the level of a competent advanced player which was clearly better than Neurogammon or any other previous backgammon program [16]. As greater computer power became available, it became possible to have longer training sessions, and to use greater depth search for real-time move decisions. An upgraded version of TD-Gammon, version 2.1, which was trained for 1.5 million games and used 2-ply search, reached the level of a top-flight expert, clearly competitive with the world's best human players [27, 29]. It was interesting to note that many of the program's move decisions differed from traditional human strategies. Some of these differences were merely technical errors, while others turned out to be genuine innovations that actually improved on the way humans played. As a result, humans began carefully studying the program's evaluations and rollouts (a Monte Carlo analysis procedure described in Section 4.2), and began to change their concepts and strategies. After analysis of thousands of positions, new heuristic principles were formulated which accounted for the new data.

This trend of human experts learning from the machine was significantly accelerated when several other researchers were able to replicate the success of TD-Gammon with

self-teaching neural nets. Two such efforts, by Fredrik Dahl and Olivier Egger, have led to the creation of commercial PC programs called Jellyfish and Snowie, respectively. These programs play at or better than world-class level and enable the user to obtain neural net evaluations or rollouts for any position. As a result, the new knowledge generated by the neural nets has been widely disseminated, and the overall level of play in backgammon tournaments has greatly improved in recent years. Kit Woolsey described some of the changes in human strategies as follows [29]:

> "Some of the previously believed concepts about backgammon were overturned. The wild slotting style of the late 1970's and 1980's was, if the neural nets were to be believed, more costly than previously thought. The race was found to be very important, and many plays were based on racing potential. Purity was found to have been overrated, while ugly attacking plays proved to be stronger than expected. The style of the average good player drifted toward these new concepts. Of course, one does wonder if these results from the bots are somewhat self-fulfilling prophecies. Could it be that the bots prefers blitzes and races to priming games and back games because it plays them better? The jury is still out on that topic."

This paper describes some of the programming issues in using self-teaching neural network technology to achieve a world-class program. To some extent, these issues have already been described in previous papers on TD-Gammon. This paper describes for the first time issues in programming n-ply search for move decisions, and in programming an algorithm for making doubling cube decisions, based on neural net evaluations.

2. Complexity in the game of backgammon

Backgammon is an ancient [1] two-player game that is played on an effectively one-dimensional track. The standard opening board configuration is illustrated in Fig. 1. The players take turns rolling dice and moving their checkers in opposite directions along the track as allowed by the dice roll. The first player to move all her pieces (commonly called "checkers" or "men") all the way forward and off the end of the board is the winner. In addition, the player wins double the normal stake if the opponent has not taken any checkers off; this is called winning a "gammon". It is also possible to win a triple-stake "backgammon" if the opponent has not taken any checkers off and has checkers in the farmost quadrant; however, this rarely occurs in practice.

The one-dimensional racing nature of the game is made considerably more complex by two additional factors. First, it is possible to land on, or "hit", a single opponent checker (called a "blot") and send it all the way back to the far end of the board. The blot must then re-enter the board before other checkers can be moved. Second, it is possible to form blocking structures that impede the forward progress of the opponent checkers. These two additional ingredients lead to a number of subtle and complex expert strategies [10,15].

[1] Precursors to the modern game existed in Egypt and Mesopotamia, possibly as much as five thousand years ago [7].

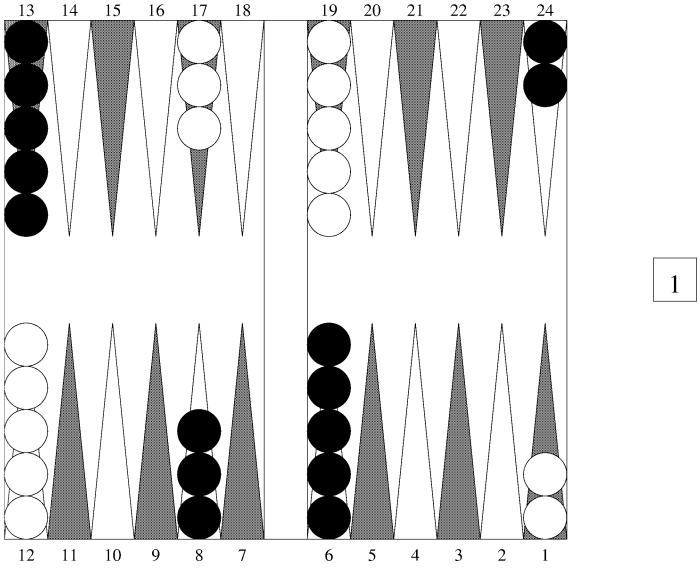

Fig. 1. Illustration of the normal opening position in backgammon. Black checkers move counter-clockwise in the direction of decreasing point numbers. White checkers move clockwise in the direction of increasing point numbers.

Additional complexity is introduced through the use of a "doubling cube" through which either player can offer to double the stakes of the game. If the opponent accepts the double, he gets the exclusive right to make the next double, while if he declines, he forfeits the current stake. Hence, the total number of points won at the end of a game is given by the current value of the doubling cube multiplied by 1 for a regular win (or for a declined double), 2 for a gammon, and 3 for a backgammon.

Programming a computer to play high-level backgammon has been found to be a rather difficult undertaking. One can't solve the full game exactly due to the enormous size of the state space (estimated at over 10^{20} states), although it has been solved exactly for a limited number of checkers (up to 3 checkers per side), and for certain no-contact endgame situations. Furthermore, the brute-force methodology of deep searches, which has worked so well in chess, checkers and Othello, is not feasible due to the high branching ratio resulting from the probabilistic dice rolls. At each ply there are 21 dice combinations possible, with an average of about 20 legal moves per dice combination, resulting in a branching ratio of several hundred per ply. This is much larger than in checkers and chess (typical branching ratios quoted for these games are 8–10 for checkers and 30–40 for chess), and too large to reach significant depth even on the fastest available supercomputers.

In the absence of exact tables and deep searches, computer backgammon programs must rely on heuristic positional judgement. The traditional approach to this in backgammon and in other games has been to work closely with human experts, over a long period of time, to design a heuristic evaluation function that mimics as closely as possible the positional knowledge and judgement of the experts [3]. There are several problems with

such an approach. First, there may be a large number of features required, and it's very difficult to articulate and code up all the useful features. Second, the features may interact with each other in complex and unanticipated ways. Third, there is no principled way to assign the correct weights for features or combinations of features. Finally, when doing knowledge engineering of human expert judgement, some of the expertise being emulated may be erroneous. As human knowledge and understanding of a game increases, the concepts employed by experts, and the weightings associated with those concepts, undergo continual change. This has been especially true in Othello and in backgammon, where over the last 20 years, there has been a substantial revision in the way experts evaluate positions. Many strongly-held beliefs of the past, that were held with near unanimity among experts, are now believed equally strongly to be quite wrong. In view of this, programmers are not exactly on firm ground in accepting current expert opinions at face value.

In the following section, we shall see that TD-Gammon represents a radically different approach toward developing a program capable of sophisticated positional judgement. Rather than trying to imitate humans, TD-Gammon develops its own sense of positional judgement by learning from experience in playing against itself. While it may seem that forgoing the tutelage of human masters places TD-Gammon at a disadvantage, it is also liberating in the sense that the program is not hindered by human biases or prejudices that may be erroneous or unreliable.

3. TD-Gammon's learning methodology

We now present a brief summary of the TD backgammon learning system. For more details, the reader is referred to [26]. A fairly detailed description of both the TD(λ) learning procedure and the TD-Gammon application is also contained in [23]. At the heart of TD-Gammon is a neural network that utilizes a standard multilayer perceptron (MLP) architecture, identical to that used in backpropagation learning [17]. The neural net may be thought of as a generic nonlinear function approximator. Given sufficient training data and sufficiently many hidden units, MLPs have been shown to be able to approximate any nonlinear function to arbitrary accuracy [6]. Furthermore, MLPs are known to have a robust capability of generalization from training cases to test cases that were not included in the training data.

The training procedure for TD-Gammon is as follows: the network observes a sequence of board positions starting at the opening position and ending in a terminal position characterized by one side having removed all its checkers. The board positions are fed sequentially as input vectors $x_1, x_2, \ldots, x_f$ to the neural network, encoded using a representation scheme that is described below. Each time step in the sequence corresponds to a move made by one side, i.e., a "ply" or a "half-move" in game-playing terminology. For each input pattern x_t there is a neural network output vector Y_t indicating the neural network's estimate of expected outcome for pattern x_t. For this system, Y_t is a four-component vector corresponding to the four possible outcomes of either White or Black winning either a normal win or a gammon. (Due to the extreme rarity of occurrence, triple-

value backgammons were not represented.) At each time step, the TD(λ) algorithm is applied to change the network's weights. The formula for the weight change is as follows:

$$w_{t+1} - w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k, \tag{1}$$

where α is a small constant (commonly thought of as a "learning rate" parameter), w is the vector of weights that parameterizes the network, and $\nabla_w Y_k$ is the gradient of network output with respect to weights. (Note that Eq. (1) expresses the weight change due to a single output unit. In cases where there are multiple output units, the right-hand side of Eq. (1) should be modified by summing over each individual output unit.)

The quantity λ is a heuristic parameter controlling the temporal credit assignment of how an error detected at a given time step feeds back to correct previous estimates. When $\lambda = 0$, no feedback occurs beyond the current time step, while when $\lambda = 1$, the error feeds back without decay arbitrarily far in time. Intermediate values of λ provide a smooth way to interpolate between these two limiting cases. Since there are no theoretical guidelines for choosing an optimal value of λ for a given nonlinear function approximator, one typically has to experiment with a range of values. Empirically, it was found with TD-Gammon that small-to-moderate values of λ gave about equally good asymptotic performance, whereas the performance degraded for large values of λ close to 1. In the initial experiments reported in [26] a value of $\lambda = 0.7$ was used. Subsequent development of TD-Gammon mostly used $\lambda = 0$: while this doesn't give a noticeable performance advantage compared to small nonzero λ values, it does have the merit of requiring about a factor of two less computation per time step.

At the end of each game, a final reward signal z (containing four components as described previously) is given, based on the outcome of the game. Once again equation 1 is used to change the weights, except that the difference $(z - Y_f)$ is used instead of $(Y_{t+1} - Y_t)$. Under these training conditions, we interpret the trained network's output as an estimate of expected outcome, or "equity" of the position. This is a natural interpretation which is exact in cases where TD(λ) has been proven to converge.

In the preliminary experiments of [26], the input representation only encoded the raw board information (the number of White or Black checkers at each location), and did not utilize any additional pre-computed features relevant to good play, such as the strength of a blockade or probability of being hit. A truncated unary encoding scheme was used for the raw board description. This required no great cleverness, as unary encodings are commonly used by neural net practitioners to encode integer data, and the truncation was imposed primarily to economize on the total number of input units. These experiments were "knowledge-free" in the sense that no knowledge of expert concepts or strategies was built in at the start of learning, nor did the neural net observe any expert move decisions during training. In subsequent experiments, a set of hand-crafted features (the same set used by Neurogammon) was added to the representation, resulting in higher overall performance, as detailed in the following section.

During training, the neural network itself is used to select moves for both sides. At each time step during the course of a game, the neural network scores every possible legal move. The move that is selected is then the move with maximum expected outcome for the side

making the move. In other words, the neural network is learning from the results of playing against itself. This self-play training paradigm is used even at the start of learning, when the network's weights are random, and hence its initial strategy is a random strategy. *A priori*, this methodology would appear unlikely to produce any sensible learning, because random strategy is exceedingly bad, and because the games end up taking an incredibly long time: with random play on both sides, games often last several hundred or even several thousand time steps. In contrast, in normal human play games usually last on the order of 50–60 time steps.

4. Results of training: TD-Gammon's move decision performance

The rather surprising finding of the experiments described in the previous section was that a substantial amount of learning actually took place, even in the zero initial knowledge experiments utilizing a raw board encoding. A sample curve illustrating the progress of learning is shown in Fig. 2. Performance is measured by periodic benchmarking of expected equity against a fixed opponent, Sun Microsystems' Gammontool program. Note that in this figure and throughout the paper, units of equity are expected points per game (ppg) won or lost. We can see in Fig. 2 that the initial random strategy loses nearly every game against Gammontool, and nearly every loss is a double-value gammon. As self-play training begins, we see that there is rapid initial learning: during the first few thousand training games, the network learns a number of elementary principles, such as hitting the opponent, playing safe, and building new points. More sophisticated context-sensitive

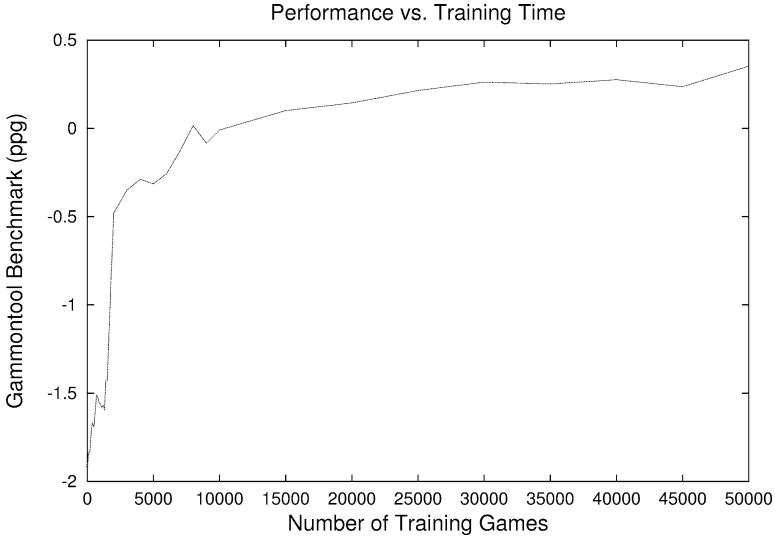

Fig. 2. A sample learning curve of one of the original nets of [26], containing 10 hidden units, showing playing strength as a function of the number of self-play training games. Performance is measured by expected points per game (ppg) won or lost against a benchmark opponent (Sun Microsystems' Gammontool program).

concepts (e.g., slotting home board points in certain situations but not in others) emerged later, after several tens of thousands of training games. The end of learning is characterized by a long slow asymptote to peak performance, which ends up being significantly better than Gammontool.

Perhaps the most encouraging finding was good scaling behavior, in the sense that as the size of the network and amount of training experience increased, substantial improvements in performance were observed. The largest network examined in the raw-encoding experiments had 40 hidden units, and its performance appeared to saturate after about 200,000 games. This network achieved a strong intermediate level of play approximately equal to Neurogammon. An examination of the input-to-hidden weights in this network revealed interesting spatially organized patterns of positive and negative weights, roughly corresponding to what a knowledge engineer might call useful features for game play [26]. Thus the neural networks appeared to be capable of automatic "feature discovery," one of the long-standing goals of game learning research since the time of Samuel.

Since TD-trained networks with a raw input encoding were able to achieve parity with Neurogammon, it was hoped that by adding Neurogammon's hand-designed features to the raw encoding, the TD nets might then be able to surpass Neurogammon. This was indeed found to be the case: the TD nets with the additional features, which form the basis of version 1.0 and subsequent versions of TD-Gammon, have greatly surpassed Neurogammon and all other previous computer programs. The improvement due to the additional features depends on the number of hidden units: a network without hidden units might improve ∼0.5 ppg, while a large net with many hidden units might improve ∼0.2 ppg.

Note that no further tinkering with the definition and encoding of features was performed as TD-Gammon was developed: the exact same features from Neurogammon were retained. It is quite likely that performance improvements could have been obtained by further refining the features based on the observed problems and weaknesses of TD learning. Indeed, it is common practice in machine learning to use knowledge engineering as a way of patching up the deficiencies of learning algorithms. However, it is this author's firm opinion, based on much experience, that this provides only short-term benefit and is dangerously likely to turn out to be a waste of time in the long run. Rather than devoting time and effort to covering up the flaws of existing learning algorithms, the ultimate goal of machine learning research should be to develop better learning algorithms that have no such flaws in the first place. As an example, the supervised learning procedure used in Neurogammon was seriously flawed in that it failed to learn the expected outcome of positions, and it failed to adequately take into account the opponent configuration in making move decisions. Much effort was expended to try to compensate for these deficiencies through clever feature design. However, when the vastly superior TD learning method was found to have no such deficiencies, this effort was revealed to be superfluous. Several of the features in the Neurogammon feature set probably could be deleted from TD-Gammon without harming its performance.

4.1. Move decisions using n-ply search

One important factor in TD-Gammon's piece movement performance, which has not received much attention in prior papers, is the ability to perform shallow-lookahead searches. Initially, the real-time move decisions of version 1.0 used simple 1-ply search, in which every top-level move is scored by the neural net, and the highest-scoring move is selected. After about 1–2 years of software and hardware speedups, versions 2.0 and 2.1 were capable of 2-ply search. The 2-ply search algorithm works as follows: First, an initial 1-ply analysis is performed and unpromising candidates are pruned based on the 1-ply score. (This is commonly known as forward pruning.) Then, the remaining top-level candidates are expanded by an additional ply. The 1-ply expansion of the surviving candidates involves making a 1-ply move decision for each of the opponent's 21 possible dice rolls, and computing a probability-weighted average score (weighting non-doubles twice as much as doubles) for each of the resulting states.

Versions 3.0 and 3.1 (the current version) are capable of a simplified 3-ply search. This is similar to the 2-ply search described above, except that a depth-2 expansion of the top level moves is performed, rather than a depth-1 expansion. The depth-2 expansion consists first doing a depth-1 expansion of the 21 dice rolls as above, selecting a move for each dice roll, and then doing an additional depth-1 expansion of the 21 followup dice rolls. In other words, a total of 441 two-roll sequences are examined, in which a 1-ply move decision is made by each side, and the score backed up to the top-level move is the probability-weighted average score of the 441 resulting successor states. This gives a huge speed advantage over full-width minimax backup, while still producing a significant boost in move quality relative to 2-ply search.

Version 3.1 of TD-Gammon contains 160 hidden units and about 50,000 floating-point weights, and was trained for over 6 million self-play games. With extensive code optimization and extensive use of pruning, it averaged about 10–12 seconds per move decision at the 1998 AAAI Hall of Champions exhibit, running on a 400 MHz Pentium II processor.

4.2. Assessing performance vs. human experts

Several methods have been used to assess the quality of TD-Gammon's move decisions relative to those of human experts. Each version of the program has typically played several dozen games against top humans; results have been quoted in previous papers. One can get an idea of the program's strength from both the outcome statistics of the games, and from the masters' play-by-play analysis of the computer's decisions. The main problem with this method is that play against humans is slow, and it is infeasible to play the several thousand games that would be required for a statistically definitive result.

Probably the most meaningful way to measure human vs. computer performance is to perform an offline "rollout" analysis of the move decisions a match between the two. A rollout is a Monte Carlo evaluation of a position in which the computer plays a position to completion many times (typically thousands of trials), using different random dice sequences in each trial. The rollout score is the average outcome obtained in each of the trials. To analyze a recorded move decision, one rolls out each candidate move, and checks

whether the recorded move obtained the highest rollout score. If so, it is deemed to be "correct", and if not, an equity loss is assigned based on the score difference between the highest-scoring move and the recorded move. Rollout analysis of move decisions, while not perfect, has been found to be extraordinarily accurate, even if the program performing the rollouts is fallible. This is due to two factors: first, for most normal backgammon positions, a program playing both sides of a position will tend to lose roughly equal amounts of equity for both sides, and thus the equity losses will tend to cancel out. Second, any systematic errors in the rollout scores of sibling top-level moves are likely to be highly correlated, since the positions are nearly identical, and would thus cancel out in determining the best move.

If there are least a few dozen games in a match, this should provide enough data to give a clear indication of the relative skill levels of the players. One might be concerned that rollouts performed by "bots" could be biased against humans. However, it appears that if there are any such biases they are likely to be small, and in any case, if there are any doubts about a rollout's accuracy, one can always redo the rollouts using a stronger player. Doing full rollouts of every decision in a long match can require a prohibitive amount of CPU time. Fortunately, it is also possible to do truncated rollouts, in which a fixed number of moves are made from the starting position, and the neural net equity estimate of the final position is recorded. Truncated rollouts are potentially much faster than full rollouts, while only giving up a small amount of accuracy in the results.

Truncated rollout analysis (depth-11, min. 3000 trials) has recently been performed for two of TD-Gammon's longer matches with top humans: the 40-game 1993 match between two-time World Champion Bill Robertie and version 2.1, and the 100-game 1998 AAAI Hall of Champions exhibition match [2] between World Cup Champion Malcolm Davis and version 3.1. (Several weeks of CPU time were required to complete the analysis.) The rollouts were performed using a recently released beta version of Snowie 3.2: this is now regarded as the strongest available rollout program, and using Snowie mitigates against the possibility that TD-Gammon rollouts might be biased in favor of itself. Results are summarized in Tables 1 and 2.

One can see that, according to the rollout statistics, TD-Gammon 2.1 technically outplayed Bill Robertie in piece-movement decisions, although the results are fairly close. The results confirm impressions at the time that the two players were fairly evenly matched. Robertie had an edge in technical plays, while TD-Gammon had an edge in vague positional situations. It is of interest to note that TD-Gammon made significantly fewer large errors, or "blunders" that gave up a large amount of equity.

Between 1993 and 1998, the rollouts indicate that TD-Gammon underwent a major improvement in playing ability, while the human performance remained relatively constant. Table 2 shows a lopsided advantage of TD-Gammon 3.1 over Malcolm Davis in equity loss, number of errors and number of blunders. About 80% of the improvement can be attributed to using 3-ply search instead of 2-ply; the remainder is due to the larger neural net with greater training experience. The 3-ply search eliminates virtually all of the program's technical errors, and the program now almost never makes any large mistakes.

[2] Only the first 95 games were used for rollout analysis. In the remaining games, Davis was playing excessively conservatively to protect his match score lead, and would have been unfairly downgraded by the rollout results.

Table 1
Rollout analysis by Snowie 3.2 of the move decisions in the 1993 match between Bill Robertie and TD-Gammon 2.1. First column gives the average cumulative equity loss per game due to inferior moves. Second column gives the average number of move decisions per game classified as "errors" (inferior to the best move by at least 0.02 ppg). Third column gives the average number of move decisions per game classified as "blunders" (inferior to the best move by at least 0.08 ppg)

Snowie rollouts	Equity loss	Avg. errors/game	Avg. blunders/game
Bill Robertie	−0.188 ppg	2.12	0.47
TD-Gammon 2.1	−0.163 ppg	1.67	0.20

Table 2
Rollout analysis by Snowie 3.2 of the move decisions in the 1998 AAAI Hall of Champions exhibition match between Malcolm Davis and TD-Gammon 3.1. Equity loss, errors and blunders are defined as in Table 1

Snowie rollouts	Equity loss	Avg. errors/game	Avg. blunders/game
Malcolm Davis	−0.183 ppg	1.85	0.48
TD-Gammon 3.1	−0.050 ppg	0.59	0.04

One would have expected Davis' 1998 performance to have surpassed Robertie's in 1993, due to the amount of theoretical progress made in the intervening years obtained by the use of neural networks as an analytical tool. Apparently this was counterbalanced by more difficult match conditions: Davis was operating in "speed-play" mode for a day and a half in an effort to complete 100 games, and there were numerous moves that appeared to be simple oversights, due to the rapidity of play. Playing at a more leisurely pace and for significant stakes, one could expect today's best humans to approach the −0.10 ppg level; however, a score of −0.05 ppg appears to be beyond human capabilities in long matches.

5. TD-Gammon's doubling algorithm

As stated previously, TD-Gammon's neural network, which estimates the cubeless equity of a position, is primarily used to make move decisions by selecting the move with the highest estimated cubeless equity. In play against humans, the neural network is also used to make doubling cube decisions, by feeding the estimated cubeless equity into a doubling formula. This formula is based on a generalization of prior theoretical work on doubling strategies published in the 1970s [9,30] and is described below.

5.1. Background on doubling theory

The approach used by backgammon experts in making doubling decisions is first to decide whether or not the opponent should accept a double. The basic rule of thumb states that a 25% cubeless chance of winning is needed in order to accept a double. At this value, the expected outcome declining the double (−1 point) equals the expected outcome

accepting the double $(0.75 \times (-2) + 0.25 \times (+2))$. Taking gammons into account, the rule states that a double can be accepted at a cubeless equity of -0.5: the equity accepting (-0.5×2) equals the equity declining (-1). In practice, doubles can be taken with less equity than this, due to the value of owning the cube: the player owning the cube can sometimes win by redoubling, whereas the player who offered the double has to then win outright.

Given the location of the opponent's take/pass indifference point, the player considering a double decides if the current position is close to crossing or has already crossed this point. If so, the player should double, and if not, the player should wait. The definition of "close" has to do with the magnitude of equity fluctuations that are likely to occur on the next 2-roll sequence. If there are sufficiently many "market-losing" sequences that cross the take/pass point, and if the magnitude by which they go past this point compensates for the bad sequences in which the player's equity deteriorates, then it is correct to double.

An important advance in doubling theory was made by Keeler and Spencer [9], who proposed the model of a binary-outcome "continuous game." In this model there is a single real variable x indicating the cubeless probability of one player winning, and at each time step x makes arbitrarily small random fluctuations. This was suggested to be a reasonable model for no-contact backgammon positions with high pip count, i.e., both players are many rolls away from bearing off all their pieces. In this model they showed that a player can accept a double with at least 20% winning chances, and a player should double right at the opponent's take/pass point. On the other hand, right at the end when the game is won or lost on the next roll, the minimal doubling point is 50%, whereas the opponent's fold point is 75%. For intermediate positions there is a smooth interpolation between these two limits, based on pip count, which was verified by computer simulation. Zadeh and Kobliska [30] worked out an analytic formula for doing the interpolation based on pip count, and verified its accuracy by more detailed and realistic computer simulations.

5.2. Generalization to multiple outcomes

In an unpublished manuscript, Tesauro [25] generalized these previous works in two ways. First, the above formalism was extended from races to more general contact positions by defining the concept of "volatility" of a position as the standard deviation in expected equity averaging over the upcoming dice rolls, and doing the interpolation between the continuous limit and the last-roll limit based on volatility. An extreme leap of faith was made that the Zadeh–Kobliska formula for the doubling threshold as a function of pip count, $T(P)$, could be converted into an equivalent function of volatility, $T(v)$, by working out the expected volatility for races of length P, and that this converted formula would also be valid for contact positions.

At the time there was no way of knowing whether this assumption was correct, as it predated the existence of TD-Gammon. In hindsight, with TD-Gammon and other strong neural net programs being capable of doing rollouts including the doubling cube, one can now accurately determine the correct doubling decisions for contact positions, and can go back and check the extent to which the converted Zadeh–Kobliska formula can also be used in contact situations. The approximation turns out to have been surprisingly accurate, except for one rarely-occuring class of positions where it gives large errors. The

type of position where this occurs is characterized by the side on roll having a moderate equity, in range of ~ 0.20–0.35, and the volatility being extremely high but not quite at the last-roll limit: $v \sim 0.50$–0.75. For racing positions with these parameters, many of these positions are redoubles, whereas for contact positions they are almost never good enough to redouble. Such contact positions often have high gammon threats for one or both sides, and if redoubled, the opponent frequently gets an efficient re-redouble on the very next roll. Due to the rarity of occurence (about once every hundred games), this "bug" in TD-Gammon's doubling algorithm persisted for several years without being detected.

The second generalization of prior work was an extension from binary outcomes to games with multiple outcomes. Ignoring backgammons, the cubeless state indicator x was extended from a scalar to a four-dimensional vector $\vec{x} = (x_1, x_2, y_1, y_2)$, where x_1 and x_2 are the probabilities of a regular or gammon win for White, and y_1 and y_2 are the probabilities of a regular or gammon win for Black. Since the probabilities must sum to 1 at all times, the fluctuations of $\vec{x}$ are constrained to lie on a 3-dimensional unit simplex defined by $x_1 + x_2 + y_1 + y_2 = 1$. The doubling points and fold points in the one-dimensional case are generalized to doubling and fold surfaces in the three-dimensional case. Obviously, in the last-roll high-volatility limit, these surfaces correspond to flat planes, representing equities of 0 and 0.5 respectively. However, in general, the surfaces may have some smooth, curved shape that would be difficult to calculate. Computing the exact shape and location of these surfaces would entail solving the steady-state diffusion equation with absorbing boundary conditions in an unusual three-dimensional geometry.

In the absence of an exact solution, Tesauro [25] proposed an approximation technique based on locating the points where the doubling surface intersects the edges of the simplex. These intersection points correspond to eliminating one of White's and one of Black's possible winning outcomes, leaving a binary game where either White wins K points or Black wins L points. There are four possible combinations of (K, L): (1,1), (1,2), (2,1) and (2,2). For each combination, we can compute the low-volatility double and fold points, using the Keeler–Spencer formalism. Having located the four intersection points, Tesauro [25] then proposed approximating the doubling and fold surfaces in the continuous limit by flat, planar surfaces that pass through the intersection points. Fortunately, the four intersection points turn out to be co-planar, so this surface is well-defined for money game play.

Having defined a low-volatility and a high-volatility doubling surface and fold surface, TD-Gammon makes doubling decisions and take-pass decisions as follows:

(1) Use the neural net to estimate the volatility v and the cubeless state vector $\vec{x} = (x_1, x_2, y_1, y_2)$ of the position.
(2) Given v, compute the interpolated doubling, redoubling, and fold surfaces using the converted Zadeh–Kobliska formulae.
(3) Determine which side of the interpolated surfaces $\vec{x}$ lies on. This determines the double, redouble, and take/pass decisions.

We also note that a similar calculation can be done of a "veto" surface, beyond which the state is too good to double, and the player should play on in the hopes of winning a gammon. Zadeh and Kobliska did not consider this case, as gammons don't occur in the types of races they examined. However, it was found that reusing the take/pass interpolation formula to also do the veto interpolation seemed to give good results in practice.

As a final remark, one can do a certain amount of hand-tuning of the doubling algorithm by multiplying the Zadeh–Kobliska interpolation coefficient by a heuristic rescaling factor. This was motivated because in the original examination of TD-Gammon 2.1, the algorithm appeared to be systematically too conservative in doubling, and much too aggressive in taking doubles. Using doubling and redoubling rescaling factors $\sim$0.9 seemed to place the program at exactly the right point where it made extremely sharp doubling decisions in line with expert judgements. For take/pass decisions, a more significant rescaling of $\sim$0.7 was used; this eliminated some of the program's bias towards bad takes. Heuristic rescaling appeared to compensate both for inaccuracies in the doubling formulae, and in systematic biases of the neural net equity estimates.

5.3. TD-Gammon's doubling performance

The doubling algorithm in TD-Gammon 2.1 used 1-ply expansion of the root nodes to make equity and volatility estimates, whereas version 3.1 used 2-ply expansion. Once again these doubling algorithms have been compared with human doubling decisions by performing Snowie rollouts of the cube decisions in the Robertie and Davis matches. The Snowie rollouts are depth-11 truncated, cubeless rollouts that apply a heuristic formula to estimate equity including the location and value of the doubling cube (i.e., "cubeful" equity) at the terminal nodes. In addition, TD-Gammon 2.1 full rollouts including the doubling cube have been performed for the Davis match. Results are presented in Tables 3 and 4. The rollouts indicate that Robertie's take/pass decisions were superb, and somewhat better than TD-Gammon's. However, TD-Gammon was clearly better in double/no double decisions: several of Robertie's doubling decisions were extremely conservative and would almost certainly be regarded by any top expert as large errors.

In the Hall of Champions match, the Snowie and TD-Gammon rollouts indicate that TD-Gammon had a slight edge in doubling decisions, and a larger edge in take/pass decisions. Davis was clearly better than Robertie in doubling decisions, whereas Robertie did better in take/pass decisions. TD-Gammon 3.1 was clearly better than version 2.1 in take/pass decisions, while it appears to have gotten worse in doubling decisions. This was due to one singular position of the type mentioned previously where the Zadeh–Kobliska formula breaks down. TD-Gammon's redouble from 4 to 8 in this one position accounted for about half its total error in the entire 100-game session. Afterwards, a modification of the Zadeh–Kobliska formula was implemented which avoids this problem and provides a much better fit to rollout data. As a result, it appears that TD-Gammon is now capable of scoring ~ -0.008 ppg in double/no double decisions. If correct, this would most likely indicate

Table 3
Rollout analysis by Snowie 3.2 (depth-11 truncated) of the cube action in the 1993 match between Bill Robertie and TD-Gammon 2.1

Snowie rollouts	BR equity loss	TD equity loss
Double decisions	−0.081 ppg	−0.013 ppg
Take/pass decisions	−0.007 ppg	−0.010 ppg

Table 4
Rollout analysis of the cube action in the 1998 Hall of Champions match between Malcolm Davis and TD-Gammon 3.1. First set of figures are based on Snowie 3.2 depth-11 truncated rollouts. Second set of figures are TD-Gammon 2.1 full rollouts including the doubling cube

Snowie rollouts	MD equity loss	TD equity loss
Double decisions	−0.031 ppg	−0.020 ppg
Take/pass decisions	−0.026 ppg	−0.005 ppg
TD-Gammon rollouts	MD equity loss	TD equity loss
Double decisions	−0.022 ppg	−0.015 ppg
Take/pass decisions	−0.026 ppg	−0.002 ppg

a slight edge over today's top humans, who would be hard pressed to reach the −0.01 ppg level in long matches.

In summary, it appears that TD-Gammon's doubling algorithm holds at least a slight advantage over world-class humans. In future research, further improvements might be obtained by utilizing a learning approach to doubling strategy. Certainly the rescaling factors and the threshold surfaces as a function of volatility could be learned by fitting to rollout data. However, a more principled and probably superior approach would be to base doubling decisions on intrinsically cubeful equity estimates, rather than plugging cubeless estimates into a heuristic formula. One method of approximating cubeful equities, which was incorporated in the latest version of Snowie, was developed by Janowski [8]. An alternative table-based approach for endgames was studied by Buro [4]. Ideally the neural net self-play training should include the doubling cube and allow the net to learn to make cubeful equity estimates. This would allow doubling decisions to be made directly by the neural net, and would also confer a slight additional benefit of being able to make checker plays taking the state of the cube into account, rather than just making the best cubeless play.

6. Conclusion

The combination of neural network function approximation and self-play learning using TD(λ) turned out to have worked much better than one could have expected for backgammon. Primitive neural nets with only a raw board input description are able to train themselves to at least a strong intermediate level of play. Adding a set of hand-designed features to the neural net's input representation, encoding concepts like blockade strength and hit probability, increases the performance to expert level. Finally, by adding a shallow search capability for real-time move decisions, a level of play is reached which by all indications is beyond current human capabilities. It was also surprising to find that, even though the doubling cube was not included in the self-play training, an excellent doubling algorithm could be obtained by feeding the neural net's cubeless equity estimates into a heuristic doubling formula. The latest evidence now suggests that TD-Gammon has a clear

advantage over top humans in piece movement decisions, and a slight advantage in cube decisions. This assessment is not seriously disputed by human experts. Malcolm Davis, for example, currently estimates that a top human player would be an underdog against any of the top neural net programs by about a tenth of a point per game.

Humans are continuing to improve their level of play by using neural net programs as an analytic tool and as a sparring partner. However, prospects for further improvement of the programs are also good, if for no other reason than the inexorable increase in computer power due to Moore's Law. This will enable more extensive training of larger neural nets, and will also allow search depths beyond 3-ply. The next significant improvement in real-time search capability will probably take the form of Monte Carlo search using truncated rollouts. This was recently studied by Tesauro and Galperin [28]; results suggest that a real-time rollout player would be 5–6 times more accurate than its base 1-ply player, and twice as accurate as the corresponding 3-ply player. While a supercomputer is currently needed to perform the rollouts in real time, one can easily envision this becoming feasible on a desktop machine in the next few years.

Beyond any specific performance achievements in the backgammon application, the larger significance of TD-Gammon is that it shows that reinforcement learning from self-play is a viable method for learning complex tasks to an extent previously unrealized by AI and machine learning researchers. Prior to TD-Gammon, it's fair to say that there had been no significant real-world applications of reinforcement learning. As a result of TD-Gammon's success, there has been much renewed interest in applying reinforcement learning in numerous real-world problem domains, and in expanding our theoretical understanding of such methods. Some of the successful applications inspired by TD-Gammon include: elevator dispatch [5], job-shop scheduling for the NASA Space Shuttle [31], cell-phone channel assignment [21], assembly line optimization and production scheduling [11,20], financial trading systems and portfolio management [13], and call admission and routing in telecommunications networks [12].

Some researchers also believe that temporal difference learning offers the hope of automated tuning of evaluation functions in many other high-performance game-playing programs [19]. As a result, there have been several applications of TD learning to other two-player board games such as Othello, Go and chess. While there has been a measure of success in these games, it hasn't been quite at the level obtained for backgammon. Amongst these other games, probably the most significant achievement of TD learning was obtained by a chess program called KnightCap, which used an extension of TD(λ) called TD-Leaf [2]. KnightCap's learning resulted in an improvement of several hundred rating points, leading to an expert rating on an internet chess server. It is of interest to note that, instead of self-play training, KnightCap trained by play against human opponents. The authors report that the program attracted progressively stronger human opposition as its rating improved, and this was essential to the success of learning.

A possible key difference between backgammon and the above-mentioned games is its intrinsic non-deterministic element due to random dice rolls. The randomness appears to have at least two beneficial effects for self-play learning. First, it provides a natural and automatic mechanism for "exploration" of a wide variety of different types of positions. Exploration is vital for reinforcement learning to work well. While exploration can be externally imposed in a deterministic game, it's not clear what would be the best way

of doing so. Second, in backgammon the game-theoretic optimal value function is a real-valued function with a great deal of smoothness and continuity, in the sense that a small change in position leads to a small change in expected outcome. Such a function is presumably easier to learn than the discrete (win, lose, draw) value functions of deterministic games, which contain numerous discontinuities where a small change in position can make a huge difference in its game-theoretic value.

In conclusion, while self-teaching neural nets have turned out to be a useful tool for programming high-performance backgammon, the discovery of this fact was not at all motivated by any performance or engineering goals. Indeed, the original expectation was that random neural nets with no built-in knowledge would be exceedingly unlikely to learn anything sensible simply by playing against themselves. However, out of simple curiosity to explore what the capabilities of TD(λ) might be, the experiments of [26] were performed and surprising results obtained. Now that the engineering goal of world-class play has been achieved in numerous games like checkers, chess, Othello, Scrabble, and backgammon [19], perhaps there will be more exploratory efforts in computer games research that study new and intriguing approaches to machine learning, and are not motivated and judged strictly on competitive performance goals. Understanding how machines may generally learn intelligent concepts and strategies in a complex environment is a worthwhile undertaking in its own right, regardless of how learning fares competitively against other methods. If used properly, the clear performance measures in computer games can measure progress in the development of learning algorithms, whereas a short-sighted attitude would be to simply dismiss any learning algorithm that failed to outperform the best competing technique on a given task. [3]

An example of exploratory research that merits further investigation is the recent work of Pollack and Blair [14] on HC-Gammon, a neural net backgammon player that evolves by random mutation and self-play test. That this method works at all is certainly surprising. HC-Gammon is both fascinating and frustrating in that it is definitely capable of learning linear structure, but unlike TD learning it appears to be incapable of extracting nonlinear structure. If correct, this would pose a serious limitation, equivalent to a backprop net being unable to learn XOR or any other high-order predicate. Determining the source of this apparent limitation, and how to overcome it, would constitute progress in the understanding and practice of evolutionary methods for training neural networks.

Three types of games seem promising for further exploratory machine learning studies. First, there are a class of games such as Connect-4 and Hypergammon (3-checker backgammon) that have been solved exactly [1], yet are challenging tasks for learning heuristic evaluation functions. Having access to the exact optimal solution for a game would greatly facilitate the assessment of the quality of learning. Second, there is the outstanding challenge offered by the game of Go. Current game-programming techniques all appear to be inadequate for developing high-performance Go programs, so there is ample motivation and opportunity to explore a variety of novel techniques. Finally, there are now opportunities to extend games research from classic two-player perfect-

[3] In the late 1980s, certain extremely famous senior scientists expressed the opinion that machine learning research in backgammon was a "failure" unless it outperformed Berliner's BKG program. Presumably they would have rejected publication of [26] since the reported performance did not match BKG's playing ability.

information games to games with more realistic characteristics, such as many players, hidden or noisy state information, continuous states and actions, and asynchronous actions and events taking place in real time. Games ranging from card games such as poker and bridge, to video games such as Doom and Quake, to economic games such as bidding and trading in auctions and financial markets, all incorporate such realistic aspects. In order for machine learning algorithms to work well in these domains, they will have to address issues that lie beyond prior studies of TD learning in games. Exploring new learning algorithms in these domains may motivate further progress in machine learning theory, and may also lead to more direct and immediate applications in general real-world problem domains.

Acknowledgement

The author thanks Olivier Egger for providing a beta version of Snowie 3.2 used to perform the rollout analysis.

References

[1] L.V. Allis, A knowledge-based approach of Connect-Four. The game is solved: White wins, M.Sc. Thesis, Faculty of Mathematics and Computer Science, Free University of Amsterdam, Amsterdam, 1988.
[2] J. Baxter, A. Tridgell, L. Weaver, KnightCap: A chess program that learns by combining TD(λ) with minimax search, in: Proc. ICML-98, Madison, WI, 1998, pp. 28–36.
[3] H. Berliner, Computer backgammon, Scientific American 243 (1) (1980) 64–72.
[4] M. Buro, Efficient approximation of backgammon race equities, ICCA J. 22 (3) (1999) 133–142.
[5] R.H. Crites, A.G. Barto, Improving elevator performance using reinforcement learning, in: D. Touretzky et al. (Eds.), Advances in Neural Information Processing Systems, Vol. 8, MIT Press, Cambridge, MA, 1996, pp. 1017–1023.
[6] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (1989) 359–366.
[7] O. Jacoby, J.R. Crawford, The Backgammon Book, Bantam Books, New York, 1970.
[8] R. Janowski, Take-points in money games, On-line article available at: http://www.msoworld.com/mindzine/news/classic/bg/cubeformulae.html(1993).
[9] E.B. Keeler, J. Spencer, Optimal doubling in backgammon, Oper. Res. 23 (1975) 1063–1071.
[10] P. Magriel, Backgammon, Times Books, New York, 1976.
[11] S. Mahadevan, G. Theocharous, Optimizing production manufacturing using reinforcement learning, in: Proc. 11th International FLAIRS Conference, AAAI Press, Menlo Park, CA, 1998, pp. 372–377.
[12] P. Marbach, O. Mihatsch, J.N. Tsitsiklis, Call admission control and routing in integrated service networks using neuro-dynamic programming, IEEE J. Selected Areas in Communications 18 (2) (2000) 197–208.
[13] J. Moody, M. Saffell, Y. Liao, L. Wu, Reinforcement learning for trading systems and portfolios, in: A.N. Refenes, N. Burgess, J. Moody (Eds.), Decision Technologies for Computational Finance: Proceedings of the London Conference, Kluwer Financial Publishing, 1998.
[14] J.B. Pollack, A.D. Blair, Co-evolution in the successful learning of backgammon strategy, Machine Learning 32 (1998) 225–240.
[15] B. Robertie, Advanced Backgammon (Vols. 1 and 2), The Gammon Press, Arlington, MA, 1991.
[16] B. Robertie, Carbon versus silicon: Matching wits with TD-Gammon, Inside Backgammon 2 (2) (1992) 14–22.
[17] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representation by error propagation, in: D. Rumelhart, J. McClelland (Eds.), Parallel Distributed Processing, Vol. 1, MIT Press, Cambridge MA, 1986.
[18] A. Samuel, Some studies in machine learning using the game of checkers, IBM J. Res. Develop. 3 (1959) 210–229.

[19] J. Schaeffer, The games computers (and people) play, in: M. Zelkowitz (Ed.), Advances in Computers 50, Academic Press, New York, 2000, pp. 189–266.
[20] J.G. Schneider, J.A. Boyan, A.W. Moore, Value function based production scheduling, in: Proc. ICML-98, Madison, WI, 1998.
[21] S.P. Singh, D. Bertsekas, Reinforcement learning for dynamic channel allocation in cellular telephone systems, in: M.C. Mozer, M.I. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems, Vol. 9, MIT Press, Cambridge, MA, 1997, pp. 974–980.
[22] R.S. Sutton, Learning to predict by the methods of temporal differences, Machine Learning 3 (1988) 9–44.
[23] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
[24] G. Tesauro, Neurogammon wins computer olympiad, Neural Comput. 1 (1989) 321–323.
[25] G. Tesauro, Optimal doubling in multi-outcome probabilistic games, IBM Research, Unpublished manuscript (1990).
[26] G. Tesauro, Practical issues in temporal difference learning, Machine Learning 8 (1992) 257–277.
[27] G. Tesauro, Temporal difference learning and TD-Gammon, Comm. ACM 38 (3) (1995) 58–68, HTML version at http://www.research.ibm.com/massive/tdl.html.
[28] G. Tesauro, G.R. Galperin, On-line policy improvement using Monte-Carlo search, in: M.C. Mozer, M.I. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems, Vol. 9, MIT Press, Cambridge, MA, 1997, pp. 1068–1074.
[29] K. Woolsey, Computers and rollouts, On-line article available at www.gammonline.com, 2000.
[30] N. Zadeh, G. Kobliska, On optimal doubling in backgammon, Management Sci. 23 (1977) 853–858.
[31] W. Zhang, T.G. Dietterich, High-performance job-shop scheduling with a time-delay TD(λ) network, in: D. Touretzky et al. (Eds.), Advances in Neural Information Processing Systems, Vol. 8, MIT Press, Cambridge, MA, 1996, pp. 1024–1030.