ELSEVIER

Theoretical Computer Science 252 (2001) 105–119

Theoretical Computer Science

www.elsevier.com/locate/tcs

Temporal difference learning applied to game playing and the results of application to shogi

Donald F. Beal *, Martin C. Smith

Department of Computer Science, Queen Mary and Westfield College, University of London, Mile End Road, London E1 4NS, UK

Abstract

This paper describes the application of temporal difference (TD) learning to minimax searches in general, and presents results from shogi. TD learning is used to adjust the weights for evaluation features over the course of a series of games, starting from arbitrary initial values. For some games, to obtain weights accurate enough for high-performance play will require the TD learning phase to make use of minimax searches. A theoretical description of TD applied to minimax search is given, and we discuss how the theoretical characteristics of the method interact with practical considerations. These include the depth of search appropriate for successful learning and the use of self-play to enable the algorithm to be independent of human knowledge. We then report on experiments that obtained values for use in shogi-playing programs. Unlike chess, shogi has no generally agreed standardized set of values for pieces, so there is more need for machine learning. We compare our machine-learnt values, obtained without any human knowledge input, with hand-crafted values. TD learning was successful in obtaining values that performed well in matches against hand-crafted values.

Keywords: Learning; Temporal difference; Minimax; Search; Shogi

1. Introduction

Temporal difference learning has a long history in game theory, starting with Samuel's 1950s work on machine learning in checkers [13]. Although similar methods were utilised in various contexts after that, it was only in 1988, after Sutton proved optimality and convergence for special cases of his TD(λ) algorithm [14], that wider interest began to be shown. In 1994, Tesauro [12] reported the creation of an international master-level backgammon program with the aid of TD(λ), and this motivated investigations of how TD(λ) might be applied to other complex games.

* Corresponding author.
E-mail addresses: don.beal@dcs.qmw.ac.uk (D.F. Beal), martins@dcs.qmw.ac.uk (M.C. Smith).

PII: S0304-3975(00)00078-5

Tesauro's program did not require look-ahead search. In backgammon it seems to be sufficient to have a good evaluation function to reach a high level of performance. This contrasts with other games, such as chess, requiring dynamic analysis of positions, and where good performance has been obtained only by programs that perform search. This paper is concerned with the application of TD(λ) to game-playing programs which need to perform minimax search during the learning phase.

The second part of this paper reports the learning of piece weights for the game of shogi using the methods described in the first part. The learning is obtained entirely from randomised self-play, without access to any form of expert knowledge. The piece values are used to evaluate the horizon nodes of a minimax search tree, and temporal difference learning is used to adjust these values over the course of a series of games.

A combination of machine-learning methods, including TD learning, was earlier used to learn chess piece values [8], and also coarse-grained piece values [4], with limited success. Beal and Smith [3] reported successful learning of piece values from self-play with no initial knowledge in the game of chess, using TD(λ) in conjunction with minimax searches.

TD(λ) learning was also used successfully by Baxter et al. [1] to improve the weights of a complex chess evaluation function consisting of positional terms as well as piece values, when playing against knowledgeable opponents. However, they found it necessary to provide piece weights as initial knowledge to obtain good performance.

Shogi is significantly different from chess, as detailed in Section 3 and there are no standardised piece values to compare against. Although human shogi players do not have fixed traditional values for shogi pieces, they do generally agree that rooks and bishops are the most powerful pieces, and that pawns have least value. Our experiment was designed to discover whether the same TD(λ) technique would perform as satisfactorily in shogi as it had in chess; whether it would yield sensible values for shogi pieces, and to examine some of the effects of search depth on learning.

2. Temporal difference learning

Temporal difference (TD) learning methods apply to multi-step prediction problems. Each prediction is a single number, derived from a formula using adjustable weights, for which the derivatives with respect to changes in weights are computable. Each pair of temporally successive predictions gives rise to a recommendation for weight changes. Sutton [14] shows that TD methods make more efficient use of their experience than conventional prediction-learning methods, converging faster and producing more accurate predictions.

Sutton's TD(λ) algorithm is based on the following formalism. Let $P_1 \ldots P_t$ be a set of temporally successive predictions, from time 1 to time t. The algorithm assumes each prediction is a function of a vector of adjustable weights w, so a prediction at time i could be written as $P_i(w)$. The algorithm further assumes that the prediction function

is differentiable so there exist partial derivatives of the prediction value with respect to each weight element. $\nabla_w P_i$ denotes the gradient, or vector of partial derivatives of prediction P at time i.

Using this notation, the weight adjustments for Sutton's TD(λ) algorithm can be expressed as

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

where α is a parameter controlling the learning rate, and λ is Sutton's recency parameter, that introduces an exponential weighting with recency of predictions occurring k steps in the past.

The process may be applied to any differentiable prediction function and any initial set of weights. Learning performance depends on λ and α, which have to be chosen appropriately for the domain.

In principle, TD(λ) weight adjustments may be made after each move, or at any arbitrary interval, but for game-playing tasks, the end of every game is a convenient point to actually alter the evaluation weights. The update rule for applying weight adjustments at the end of each game is

$$w \leftarrow w + \sum_{i=1}^{t} \Delta w_i. \tag{2}$$

2.1. Obtaining prediction probabilities from evaluation scores

To apply TD(λ) to game-playing there are some important requirements of the prediction function that need to be considered. The first of these is that the function for prediction values should integrate smoothly with end-of-game values. In particular, the value for checkmate should be close to that for being heavily ahead in the summation of evaluation terms.

Another, related, requirement is that the prediction values should approximate the utility of the game result. For games such as chess and shogi, the utility is 1 for any win, 0 for any loss. (This contrasts with games such as bridge, where winning the game brings a variable amount of reward.)

Both of these requirements can be achieved by using a squashing function, which converts from the conventional polynomial evaluation function typically used in game-playing programs, to a probability of winning. It is convenient to use a standard sigmoid squashing function. The prediction P, the probability of winning from a given position, is determined by using the function

$$S(v) = \frac{1}{1 + e^{-v}}, \tag{3}$$

where v is the evaluation value of the position. In typical games programs the evaluation function is often a polynomial formed from linearly weighted evaluation terms

$$v(x) = \sum_{i=1}^{n} w_i c_i, \tag{4}$$

where c is a vector of n evaluation terms computed for the position x. Thus, the prediction P is given by

$$P = S(v(x)). \tag{5}$$

The sigmoid function (3) has the advantage that it has a simple derivative

$$\frac{\partial S}{\partial v} = S(1 - S). \tag{6}$$

Therefore, the partial derivative of the prediction with respect to an individual weight w_i is

$$\frac{\partial S}{\partial w_i} = \frac{\partial S}{\partial v}\frac{\partial v}{\partial w_i} = S(1 - S)c_i. \tag{7}$$

The derivative of S appears in classical supervised-learning procedures as well as TD(λ). The effect of the derivative in the weight adjustment formula is that weights receive adjustment in proportion to their effect on the prediction. In other words, weights that have little influence on the prediction are adjusted less than weights to which the prediction is more sensitive.

2.2. Using the principal position, rather than the position reached in the game

An important aspect of applying TD(λ) to minimax search is selecting the correct position to use in the computation of weight adjustments. When performing minimax search to make move choices, the evaluation score from the position at the end of the principal variation (the *principal position*) is backed up to the root of the search.

Let $g_1, g_2, g_3, \ldots, g_n$ be the positions of game G. Let $h_1, h_2, h_3, \ldots, h_n$ be the principal positions identified by the minimax searches from $g_1, g_2, g_3, \ldots, g_n$.

Thus, $P_i = S(v(h_i))$.

At the end of the game, the outcome is defined by the rules of the game, thus $g_n = h_n = \{0 \mid 0.5 \mid 1\}$ where $\{0 \mid 0.5 \mid 1\}$ means one of the values 0, 0.5, or 1.

It is the value from the principal position that is the prediction of the final outcome of the game, to be compared with future values by the temporal difference method. Consequently, the computation of partial derivatives must be performed with evaluation terms calculated at the principal position h_i, not at the position in the game g_i.

2.3. The effect of depth of search on the learnt values

For games that require minimax search to achieve high standards of play, an important consideration is the time taken. Obviously, we would like to learn from as little

computational effort as possible, yet the searching during the learning phase must be deep enough to achieve sufficient quality of play to inform the learning. "Sufficient" quality of play means that when one side has the advantage, the subsequent play by that side must be able to obtain a win (not necessarily every game, but the benefit must be visible when averaged over many games).

In practice, therefore, it is necessary to determine a sufficient depth of search for learning the particular evaluation terms in the game being studied. We can expect that some minimum search depth is necessary, and that once sufficiency is reached, greater search depths will not greatly improve the learning, nor greatly improve the final values. As execution time is an exponential function of search depth, it is of great practical importance to know (at least roughly) what the minimum depth is. In Section 6 we present results from our experiments in shogi, at different search depths, intended to address this question.

By considering the formulae in this section, we may expect another depth-related learning phenomenon of largely theoretical, rather than practical, interest. As search depth increases, and concomitantly quality of play rises, we can expect smaller differences in evaluation terms to secure wins and the rewards that go with them. Thus, the weight values learnt will reflect this increasing sensitivity to evaluation terms. Accordingly, we can expect that the absolute values of weights will tend to rise as quality of play improves. This is mainly of theoretical interest, because it is only the relative values of the weights that determine move choice. Nevertheless, we report observations on this phenomenon in Section 6.

2.4. Self-play

Self-play offers potential applicability to problems where existing expertise is not available, or where the computer program may be able to go beyond the level of existing knowledge. The focus of this paper is on learning from self-play alone, with no knowledge input.

From the theoretical perspective self-play may also learn from fewer games (there are twice as many learning opportunities in each game), but self-play may never explore some portions of the game space, due to insufficient variety in move choice. In the experiments reported in Section 6, we included a randomizing element in the move decisions, to provide a wide variety of games.

3. Shogi

Shogi is the Japanese name for the Japanese version of chess. It belongs to the same family of games as western chess and Chinese-chess (Xiangqi). Throughout this paper we refer to western chess as just *chess*, and Japanese chess as *shogi*. An introduction to the rules and some basic strategies of shogi is given by Fairbairn [6] and Leggett [7]. Matsubara et al. [10] suggest shogi is an appropriate target for current game-playing

research, and discuss the similarities and differences between shogi and chess. In shogi, captured pieces are not eliminated from the game, but kept *in hand* by the capturing player, and may later be returned (*dropped*) on almost any vacant square. This greatly increases the branching factor of the game tree, and makes the game less amenable to full-width searching techniques. An additional feature of shogi is that all pieces apart from the king and gold are eligible for promotion once they reach the *promotion zone* (the last three ranks of the board). Pawns, lances, knights and silvers may all promote to golds upon entering the promotion zone, whereas rooks and bishops promote to more powerful pieces.

Choosing suitable values for shogi pieces is a problem for game programmers, as shogi experts prefer not to allocate values to the pieces. Sensible values for chess pieces are fairly widely known, but there is no generally agreed standardized set of values for shogi pieces. Hence, shogi programmers have more need for machine learning to generate material values for use in evaluation functions.

Note that in Shogi, unlike chess, the value of a piece is not the same as the change in the material balance when a piece is captured. For example, when capturing an opponent's promoted rook the change in material balance needs to take into account both the loss of the promoted rook to the opponent, and also the gaining of a rook in hand for the capturing side.

4. The Shogi-playing search engine

The experiments used a search engine derived from a conventional chess program, with an iteratively deepened search, alpha–beta pruning and a captures and promotions only quiescence search at the horizon [9]. To prevent undue search effort being expended in the quiescence search, it was limited to eight plies. (We performed some test runs with an unlimited quiescence search, and obtained essentially identical results, but at greater computational cost.) Null-move pruning [3, 5] was used to reduce the size of the search tree, and the search was made more efficient by the use of a transposition table. The evaluation function applied at the leaves of the quiescence search consisted of the material score only, with the move choice at the root being made randomly from the materially equal moves.

The 13 piece values being learnt were used by the evaluation function. The *material balance* for a position was calculated as the sum of all the values of the side to move's pieces (including pieces in hand), minus the sum of all the values of the opponent's pieces.

5. The experiments

The experiments were designed to test the application of TD(λ) to the task of learning suitable piece values for a shogi-playing program. Many learning runs were performed

to explore the behaviour of the TD(λ) method using a variety of search depths. Suitable values for the learning rate, and values for λ were determined by some preliminary test runs. All games were played using the shogi engine described in Section 4, with a main search varying in depth between one and four plies. To prevent the same games from being repeated, the move lists were randomised, resulting in a random choice being made from all tactically equal moves. This has the added benefit of ensuring a wide range of different types of position are encountered.

During each game a record is kept of the value returned by the search after each move, and the corresponding principal position. These values are converted into prediction probabilities by the squashing function given in Eq. (3), and then Eqs. (1) and (2) are used to determine adjustments to the weights at the end of each game. At the start of each run all weights were initialised to 1, so that no game-specific knowledge was being provided via the initial weights.

The experiments learn values for the pieces entirely from randomised self-play. This method has the advantage that it requires no play against well-informed opponents, nor is there any need for games played by experts to be supplied. The piece weights are learnt "from scratch", and do not need to initialised to sensible values. The only shogi-specific knowledge provided is the rules of the game. Whilst each learning run consists of several thousand games, this represents a relatively short amount of machine time, and the entire run can be completed without any external interaction.

6. Results

We present results from multiple runs using varying random seeds, and varying search depths. At each depth (1–4), five separate learning runs (a–e) were conducted. These learning runs were identical except that a different random number seed was used in each one, ensuring that completely different games were played in each. We examine the learning behaviour and the variations observed in the learnt values.

6.1. Typical weight traces

Fig. 1 shows the weight traces for main pieces for a typical learning run (c) of 6000 games at search depth 3. A decaying learning rate was used for the first half of the run, decreasing from 0.05 to 0.002. Once the learning rate reached 0.002, it remained constant for the remainder of the run. Very similar results were achieved using a fixed learning rate of 0.002, but the runs required more games to achieve stable values. The recency parameter λ from Eq. (1) was set to 0.95 after some initial experiments to determine a suitable value.

From Fig. 1, we can see that the relative ordering of the main pieces has been decided after about 4000 games, and that pieces remain in that relative order for the remainder of the run. During the last 2000 games there is still considerable drift in the values. Some random drift is to be expected as a result of the random component

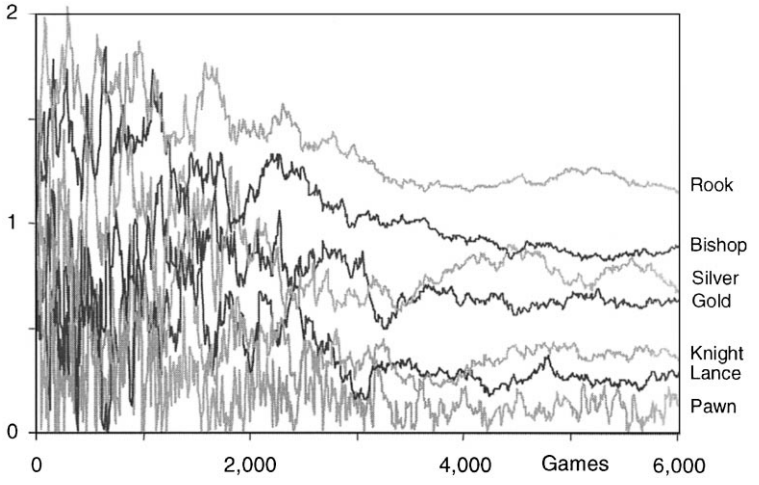

Fig. 1. Typical weight traces (main pieces) from a depth 3 run.

included in the move choice. We averaged the values over the last 2000 games in order to obtain values for testing against other weight sets. Very similar traces were obtained at all search depths, although at shallower depths, with their corresponding lower quality of play, more games were required before the values stabilised.

6.2. *Variation of learnt values with random seed*

Fig. 2 shows relative values for the seven main piece types, from each of the five learning runs conducted at search depth 3. This information is also available in numerical form in the appendix. To avoid fluctuations in the weights due to noise from the stochastic nature of the game-playing process, these values represent the average over the last 2000 games in each of the five runs.

It is the *relative* values of the pieces that governs move selection, not the *absolute* values. This enables us to conveniently compare the values from the five runs by normalising them so that the value of an unpromoted rook equals 5. (In chess, one often refers to the values of pieces in terms of pawns, e.g., "A knight is worth three pawns". In shogi, there is no such commonly used metric. However, in certain rare situations [6], the rules of shogi state that rooks are to be scored as five points each, and minor pieces as one point each. We chose to use the five-point rook score as our reference value for normalising. This choice of normalisation method is a presentation issue only, and has no effect on the experiments.)

From Fig. 2 we can see that each of the five runs has learnt the same ordering of the pieces (Pawn, Lance, Knight, Silver, Gold, Bishop and Rook). In addition, the relative magnitude of the learnt values is remarkably consistent across the five runs. This consistency across all random seeds was present at every search depth tested, as can be seen in Fig. 3 and Tables 4 and 5.

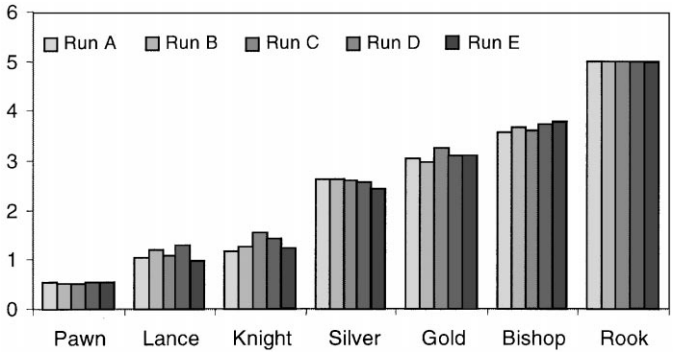

Fig. 2. Normalised learnt values for five runs (main pieces) at search depth 3.

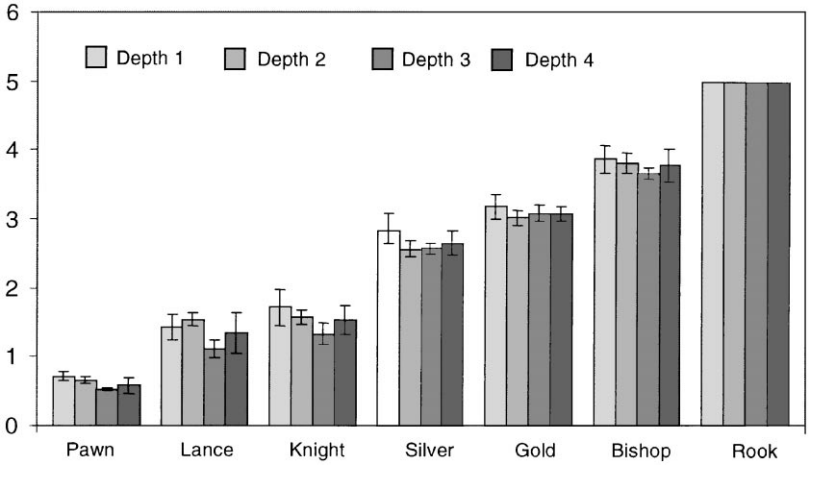

Fig. 3. Piece values at depth 1–4, normalised to rook = 5.

6.3. *Variation of learnt values with search depth*

Fig. 3 shows the average piece values learnt at each of the four depths, normalised so that rook = 5. The standard deviation from the five different random seeds is shown as a vertical line embedded in the top of each bar. This figure shows that the values learnt for the main piece types are fairly consistent across search depths, with the relative ordering of the pieces being the same in every case. The raw results used to construct this figure are given in Tables 4 and 5.

The search depths used for learning runs refer to the main search prior to the quiescence search. Thus, "depth 1" means 1 ply of main search followed by the quiescence search. We also investigated search regimes which invested even less computational effort. We performed learning runs that used no search at all, and that used 1 ply without quiescence. We found that the no-search runs failed to learn useful values, and the 1-ply-no-quiescence runs learnt much more slowly and erratically, and had not

Table 1
Average piece values (before normalisation)

	Depth	Pawn	Lance	Knight	Silver	Gold	Bishop	Rook
Main	1	0.08	0.17	0.21	0.34	0.38	0.47	0.60
	2	0.12	0.29	0.30	0.48	0.57	0.72	0.94
	3	0.13	0.28	0.33	0.64	0.77	0.92	1.25
	4	0.13	0.29	0.34	0.57	0.67	0.82	1.09
Promoted	1	0.22	0.12	0.14	0.24	—	0.94	1.08
	2	0.73	0.52	0.55	0.65	—	1.32	1.66
	3	0.70	0.32	0.50	0.23	—	1.68	2.02
	4	0.71	0.60	0.69	0.77	—	1.35	1.66

approached stable values by the end of the runs. We interpret these results as being due to the quality of play being too poor to inform the learning. When an advantage was obtained by one side, the subsequent play was not good enough to consistently convert that advantage into a win. Inspection of one or two sample games lent support to this interpretation.

The results show that the values are fairly consistent over search depths from 1-ply plus quiescence upwards.

6.4. Scaling variation with depth

Table 1 gives the average piece values (before normalisation) for each of the four depths, for both the main and promoted piece types. Golds never promote, and so have no entry in the promoted section. Comparing the depth 2 values with those from depth 4, it can be seen that the absolute values obtained using better quality of play (depth 4) are greater than from the lesser depth.[1] The same scaling variation with depth can be seen when comparing depths 1 and 3. Note that although the absolute values differ from depth 2 to 4, the relative values (which are what determine move choice) are very similar, as can be seen in Fig. 3.

The fact that the relative values vary little with an increase in search depth is encouraging for the use of this method by competitive shogi programs, which typically operate with a search depth greater than four plies. The computational cost of such searches makes learning runs of thousands of games at those depths infeasible, but these experiments show that shallower searches may well be able to produce results that are useful at deeper depths.

6.5. Matches to test the learnt values

To test the effectiveness of the learnt values in our domain, a number of matches were played between identical search engines using various different piece values. The

[1] We do not compare values from odd and even depths as it is well known that search evaluations oscillate with odd and even depths and this effect interacts with the scaling variation.

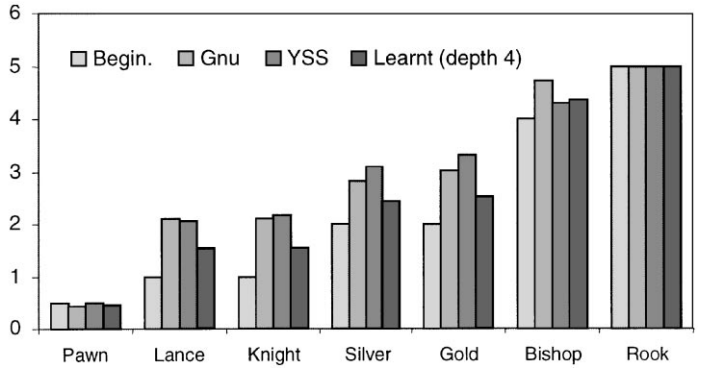

Fig. 4. Value sets tested in match play (main pieces).

search engines used were the same as used in the learning experiments, but the piece weights were fixed to a given set of piece values, and not adjusted during the match.

The matches compared the average values from all five learning runs at each depth (i.e. those in Table 1) with three value sets obtained from other sources: *Beginner*, *Gnu-derived*, and *YSS*. Fig. 4 shows the main piece values used in these sets, and for comparison purposes includes the values learnt at search depth 4. The exact values used in the matches can be found in Table 3.

The *Beginner* piece values were decided by a shogi beginner (but experienced game programmer), guided by the advice given by Leggett [7].

The *Gnu-derived* piece values were derived from those used by the widely available program *Gnu shogi* [11]. This program uses four different sets of piece values, depending on the stage of the game, as determined by various heuristics. The *Gnu* values are the average of these four sets.

The *YSS* piece values are those published on the WWW by Hiroshi Yamashita [15], author of *YSS* 7.0, winner of the 7th World Computer Shogi Championship in 1997.

The evaluation functions of both Gnu shogi and YSS also contain more sophisticated positional terms, e.g. king safety. In both programs, piece values are fundamental and typically the largest component of the overall evaluation score for a position. (Positional factors may also reward material possession indirectly. We ignored this secondary effect for these value sets.)

Each match consisted of 2000 games, alternating black and white (sente and gote). Games that ended in mate were scored as 1 point for the winning side. Games that were unfinished after 600 ply (300 moves each) were scored as $\frac{1}{2}$ a point for both sides.

The results of the matches played are given in Table 2. The learnt values from depths 1–4 consistently performed better than any of the other value sets under our test conditions, scoring 51%, 54% and 58% and 55% against the YSS value set, which was the best of the opponent value sets.

Table 2
Match results

Depth	*YSS* (%)	Gnu (%)	Beginner (%)
1	51	52	55
2	54	59	62
3	58	64	76
4	55	60	66

7. Conclusions

This paper described the application of TD learning to minimax searches, and presented results from the game of shogi. The shogi piece values were learnt from self-play without any domain-specific knowledge being supplied. Although shogi experts are traditionally reluctant to assign values to the pieces, we believe that our learnt values would be recognised by human experts as reasonable for use in a shogi program. The values learnt using various depths of search all performed well in matches under our test conditions, and the consistency of the relative values across the search depths indicates that a one-ply plus quiescence search is already sufficient to learn reasonable values. This is encouraging for the potential application of this method to the learning of weights for use by deep-searching, high-performance programs. It indicates that the learning process may be able to use much shallower, faster searches than the playing program, and thus obtain values from large numbers of training games in a reasonable time.

It should be noted that these experiments have learnt material values within a material-only evaluation function. We would expect the material values learnt to be somewhat different if the evaluation function included positional scoring terms. Also, our results are obtained using a specific set of search parameters (selectivity, quiescence details, etc). These may influence the optimum values, although we would expect changes to search parameters to have less of an effect on learnt values than additional evaluation terms. The method could be applied to any other set of search parameters, and other search engines. It is also applicable to learning an appropriate weight for positional evaluation terms, and we expect it to be useful in learning weights for more sophisticated evaluation functions in both chess and shogi.

Appendix

The exact values used in the matches are given in Table 3.
The raw results used to construct Fig. 3 are given in Tables 4 and 5.

Table 3
Piece values used in matches (normalised to rook = 5)

		Pawn	Lance	Knight	Silver	Gold	Bishop	Rook
Main	Begin.	0.50	1.00	1.00	2.00	2.00	4.00	5.00
	Gnu	0.42	2.11	2.11	2.83	3.03	4.74	5.00
	YSS	0.48	2.07	2.16	3.08	3.32	4.28	5.00
Promoted	Begin.	1.50	1.75	1.75	2.00	—	5.00	6.00
	Gnu	1.91	2.50	2.63	3.16	—	5.21	5.16
	YSS	2.02	3.03	3.08	3.22	—	5.53	6.25

Table 4
Main piece values for each of the five runs (a–e) at depths 1–4

Depth		Pawn	Lance	Knight	Silver	Gold	Bishop	Rook
1	a	0.08	0.16	0.19	0.33	0.39	0.45	0.59
	b	0.08	0.17	0.21	0.37	0.39	0.50	0.61
	c	0.08	0.16	0.23	0.34	0.41	0.49	0.60
	d	0.08	0.15	0.23	0.31	0.37	0.47	0.62
	e	0.09	0.19	0.20	0.33	0.33	0.44	0.56
2	a	0.13	0.28	0.32	0.49	0.56	0.71	0.93
	b	0.13	0.32	0.26	0.44	0.54	0.70	0.93
	c	0.12	0.29	0.31	0.49	0.57	0.72	0.95
	d	0.11	0.30	0.30	0.52	0.59	0.73	0.99
	e	0.13	0.27	0.29	0.48	0.58	0.74	0.91
3	a	0.12	0.24	0.27	0.60	0.70	0.83	1.15
	b	0.13	0.32	0.34	0.71	0.80	0.99	1.35
	c	0.12	0.26	0.38	0.63	0.79	0.87	1.21
	d	0.13	0.32	0.35	0.64	0.77	0.92	1.25
	e	0.14	0.26	0.32	0.64	0.81	0.98	1.30
4	a	0.11	0.35	0.37	0.58	0.69	0.90	1.07
	b	0.16	0.36	0.36	0.65	0.68	0.84	1.10
	c	0.14	0.32	0.38	0.57	0.71	0.86	1.14
	d	0.10	0.21	0.25	0.52	0.62	0.77	1.05
	e	0.12	0.23	0.33	0.55	0.64	0.74	1.06

Table 5
Promoted piece values for each of the five runs (a–e) at depths 1–4

Depth		Pawn	Lance	Knight	Silver	Gold	Bishop	Rook
1	a	0.19	0.09	0.20	0.23	—	0.95	1.07
	b	0.22	0.12	0.24	0.13	—	0.93	1.04
	c	0.27	0.13	0.04	0.43	—	0.97	1.15
	d	0.25	0.11	0.04	0.28	—	0.95	1.11
	e	0.17	0.13	0.15	0.11	—	0.89	1.05
2	a	0.76	0.58	0.59	0.67	—	1.39	1.60
	b	0.72	0.49	0.60	0.59	—	1.23	1.71
	c	0.74	0.54	0.58	0.67	—	1.35	1.65
	d	0.72	0.48	0.53	0.69	—	1.30	1.72
	e	0.73	0.50	0.48	0.65	—	1.33	1.63
3	a	0.72	0.35	0.54	0.59	—	1.46	1.88
	b	0.72	0.25	0.57	0.02	—	1.84	2.11
	c	0.64	0.18	0.51	0.22	—	1.53	1.93
	d	0.63	0.43	0.30	0.27	—	1.70	2.03
	e	0.82	0.38	0.56	0.06	—	1.85	2.13
4	a	0.74	0.65	0.73	0.85	—	1.43	1.74
	b	0.85	0.78	0.78	0.85	—	1.32	1.61
	c	0.75	0.80	0.79	0.80	—	1.37	1.69
	d	0.56	0.36	0.58	0.67	—	1.31	1.66
	e	0.65	0.39	0.57	0.69	—	1.31	1.57

References

[1] J. Baxter, A. Tridgell, L. Weaver, KnightCap: A chess program that learns by combining TD(lambda) with game-tree search, in: Machine Learning, Proceedings of the 15th International Conference, ICML '98, Madison, 1998, pp. 28–36.

[2] D.F. Beal, Experiments with the Null Move, in: D.F. Beal (Ed.), Advances in Computer Chess 5, Elsevier, Amsterdam, 1989, pp. 65–79.

[3] D.F. Beal, M.C. Smith, Learning piece values using temporal differences, Internat. Comp. Chess Ass. J. 20 (1997) 147–151.

[4] J. Christensen, R. Korf, A unified theory of heuristic evaluation functions and its application to learning, AAAI-86, Morgan-Kaufman, Los Altos, CA, 1986, pp. 148–152.

[5] C. Donninger, Null move and deep search: selective search heuristics for obtuse chess programs, Internat. Comp. Chess Ass. J. 16 (1993) 137–143.

[6] J. Fairbairn, Shogi for Beginners, Ishi Press International, 1989.

[7] T. Leggett, Shogi: Japan's game of strategy, Charles E. Tuttle Company, reprinted 1993, first published 1966.

[8] R. Levinson, R. Snyder, Adaptive pattern oriented chess, Proceedings of AAAI-91, Morgan-Kaufman, Los Altos, CA, 1991, pp. 601–605.

[9] T.A. Marsland, Computer Chess and Search, in: S. Shapiro (Ed.), Encyclopaedia of artificial intelligence, 2nd ed., Wiley, New York, 1992.

[10] H. Matsubara, H. Iida, R. Grimbergen, Natural developments in game research: from chess to shogi to go, Internat. Comp. Chess Ass. J. 19 (1996) 103–112.

[11] M. Mutz, Gnu Shogi, vol. 2, p. 3 (Available from many sources, including ftp://ftp.uni-passau.de/pub/local/shogi, 1994).

[12] G. Tesauro, TD-Gammon, a self-teaching backgammon program, achieves Master level play, Neural comput. 6 (1994) 215–219.

[13] A.L. Samuel, Some studies in machine learning using the game of checkers, IBM J. Res. Dev. 3 (1959) 210–229. Reprinted in E.A. Feigenbaum, J. Feldman (Eds.), Computers and Thought, McGraw-Hill, New York.
[14] R.S. Sutton, Learning to predict by the methods of temporal differences, Machine Learning 1988 (9–44).
[15] H. Yamashita, YSS: About the Data Structures and the Algorithm, published on the WWW at http://plaza15.mbn.or.jp/~yss, 1997.