Temporal Difference Approach to Playing Give-Away Checkers

Jacek Mańdziuk and Daniel Osman

Faculty of Mathematics and Information Science, Warsaw University of Technology,
Plac Politechniki 1, 00-661 Warsaw, POLAND
mandziuk@mini.pw.edu.pl, dosman@prioris.mini.pw.edu.pl

Abstract. In this paper we examine the application of temporal difference methods in learning a linear state value function approximation in a game of give-away checkers. Empirical results show that the TD(λ) algorithm can be successfully used to improve playing policy quality in this domain. Training games with strong and random opponents were considered. Results show that learning only on negative game outcomes improved performance of the learning player against strong opponents.

1 Introduction

The temporal difference algorithm TD(λ) [1] has been successfully used in many games: backgammon [2], checkers [3, 4], chess [5], go [6], othello [7] and other. It was first used by A.L. Samuel in 1959 [3] but received it's name after the work of R. Sutton [1].

In this paper we apply the TD(λ) method to the US variant of give-away checkers (GAC). The rules of GAC [8] are exactly the same as in ordinary checkers, the only difference between these two games is the goal. In GAC a player that *loses* all his pieces is considered a winner. Formally a player wins if in his turn no legal move can be made. Although computer checkers have achieved world class game play [9], there is no known GAC program able to compete with the best human players. The game at first glance may seem trivial or at least not interesting, however a closer look reveals that it may be even a harder game to play than checkers. For example a simple piece disadvantage isn't a good estimation of in-game player's performance. A player left with one king can easily be forced to eliminate all of the opponent's pieces.

2 Value Function

In order to assign values to non-terminal states $s \in S$ the following state value function approximation was used:

$$V(s, w) = a \cdot \tanh\Big(b \cdot \sum_{k=1}^{K} \omega_k \cdot \phi_k(s)\Big), \qquad a = 99,\ b = 0.027 \tag{1}$$

where $\phi_1(s), \ldots, \phi_K(s)$ are state to integer mapping functions also called basis functions or the elements of a state feature vector. $w = [\omega_1, \ldots, \omega_K]^T \in \mathbb{R}^K$ is the tunable weight vector. $a = 99$ to guarantee that $V(s, w) \in (-99; +99)$ and $b = 0.027$ so that $a \cdot \tanh(b \cdot 99) \approx 99$. The $tanh(\cdot)$ function was used only for technical reasons. Besides this the value function $V(s, w)$ can be seen as a weighted sum of basis functions $\phi_i(s)$. For terminal states $s \in T$ the values of $V(s, w)$ are +100 for win, 0 for tie and -100 for loss for any $w \in \mathbb{R}^K$.

The value function is used to assign values to leaf nodes of a fixed-depth d-ply mini-max game search tree. After that a move is executed following the best line of play found. In this case it is said that a player follows a greedy policy because at every state always the best move according to the program is made.

3 Temporal Difference Algorithm. Learning a Policy

The TD(λ) algorithm is used to modify weights. At time t the weight update vector Δw_t is computed from the following equation:

$$\Delta w_t = \alpha \cdot \delta_t \cdot e_t \tag{2}$$

where $\alpha \in (0, 1)$ is the learning step-size parameter. The second component $\delta_t = r_{t+1} + \gamma V(s_{t+1}^{(l)}, w) - V(s_t^{(l)}, w)$ represents the temporal difference in state values. r_{t+1} is the scalar reward obtained after a transition from state s_t to s_{t+1}. $\gamma \in (0; 1)$ is the discount parameter. In our experiments $\gamma = 1$ and $r_t = 0$ for all t, although a small negative value of r_t could have been used in order to promote early wins. $s_t^{(l)}$ is the principal variation leaf node obtained after performing a d-ply mini-max search starting from state s_t (the state observed by the learning player at time t). In other words $V(s_t^{(l)}, w)$ is the mini-max value of state s_t or a d-step look-ahead value of s_t. The last component of equation (2) is the eligibility vector e_t updated in the following recursive equation:

$$e_0 = 0, \qquad e_{t+1} = \nabla_w V_{t+1} + (\gamma\lambda) e_t \tag{3}$$

where $\lambda \in (0, 1)$ is the decay parameter. $\nabla_w V_k$ is the gradient of $V(s_k, w)$ relative to weights w. Formally the i-th element of this gradient equals:

$$(\nabla_w V_k)_i = \frac{\partial V(s_k, w)}{\partial w_i} = \phi_i(s_k) \quad i = 1, \ldots, K, \;\; k = 1, 2, ... \tag{4}$$

where K is the size of the weight vector and s_k is the state observed at time k. The eligibility vector holds the history of state features encountered. The elements of this vector (unless they are encountered again) decay exponentially according to the decay parameter λ. The features in e_t are thought to be significant while performing weight updates at time t. The eligibility vector is the main tool used by delayed reinforcement learning for assigning credit to past actions [10].

In TD(λ) weight updates are usually computed on-line after every learning player's move. However in this paper the learning player's weights are changed only after the game ends (off-line TD(λ)). This enables us to condition weight replacement with the final result of the game (win, loss or tie).

The weight vector determines the value function which in turn determines the player's policy $\pi : (S, \mathbb{R}^K) \rightarrow A$. Policy $\pi(s, w)$ is a function that for every state $s \in S$ and some weight vector $w \in \mathbb{R}^K$ maps an action $a \in A$. Thus tells the computer player which move to perform at state s. Our goal is to learn an optimal policy that maximizes the chance of a win (learning control). This is achieved indirectly by learning to predict the final outcome of being in state s.

4 Experiment Design and Results

The computer player that has his weights changed (trained) after every game by the $TD(\lambda)$ algorithm is called the learning player. There were 10 learning players each having it's own set of 25 opponents. All weights for players 1 and 6 where initialized with 0. The rest of the players had their vectors initialized with random numbers $r \in (-10.0, +10.0)$. Each of the 250 opponents also had their weight vector initialized with random numbers. All the opponent's weight vectors were pairwise different and were never changed during learning. In training game number i the learning player played against opponent $(i \bmod 25) + 1$. Players 1 to 5 always played using white pieces (they performed the first move). Players 6 to 10 played using black pieces. For every win the learning player received 1 point, for a tie 0.5 and for a loss 0 points.

The game tree search depth $d = 4$ mainly due to time limitations. The size of the weight vector $K = 22$. In games marked with LL (learn on loss) the learning player's weights were modified only in the case of its loss or tie. In games marked with LB (learn on both) the weights were modified after every game no matter the final outcome.

In order to check if the performance in the training phase really reflects policy improvement, a testing phase was introduced. In the testing phase one weight vector achieved by the learning player in some point in time is taken and used by the testing player to play against $1,000$ new random opponents. These opponents were not encountered earlier by the learning players. The same set of $1,000$ opponents was used in all testing phases mentioned in this paper.

4.1 Tuning α and λ

Over 300,000 initial games were played in order to observe performance with different values of parameters α and λ chosen. The results in this phase showed that applying a good pair of parameters in the right moment can substantially shorten the learning time and increase results. Finally the following scheme for decreasing α and λ was chosen: in games $1 - 2,500$: ($\alpha = 1E - 4, \lambda = 0.95$), in games $2,501 - 5,000$: ($\alpha = 2E - 5, \lambda = 0.7$), in games $5,001 - 7,500$: ($\alpha = 1E - 5, \lambda = 0.5$) and in games $7,501 - 10,000$: ($\alpha = 5E - 6, \lambda = 0.2$). All the following results are based on games played using this scheme.

Table 1. Training and test phase performance. Each result is averaged over 10 passes, one for each learning player.

(a) Training phase. Random opponents

games	LL	LB
1 - 2,500	64.9%	64.7%
2,501 - 5,000	70.5%	70.2%
5,001 - 7,500	72.0%	72.2%
7,501 - 10,000	70.8%	69.1%

(b) Training phase. Strong opponents

games	LL	LB
1 - 2,500	61.8%	52.9%
2,501 - 5,000	64.3%	54.9%
5,001 - 7,500	71.9%	51.8%
7,501 - 10,000	62.8%	52.3%

(c) Test phase. Random opponents

after game	LL	LB
7,500	70.2%	68.8%
10,000	69.8%	68.7%

(d) Test phase. Strong opponents

after game	LL	LB
7,500	72.3%	69.1%
10,000	72.9%	68.9%

4.2 Learning on 25 Random Opponents

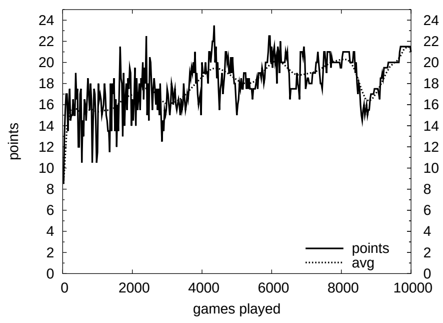
Table 1(a) presents the results of training the learning players on 25 random opponents. It can be seen that both the LL and LB methods achieved comparable results. The highest average result (72.2%) was obtained using the LB method during games 5,001-7,500. In subsequent games the performance decreased.

Results of the learning players during training were observed after every 25 games. Performance history of the best players learning with the LL and LB methods on 25 random opponents are presented in Figs. 1(a) and 1(b), resp. The best overall result was achieved by player 9-LB some time after the 6000-th game (Fig. 1(b)) and exceeded 88% of the possible maximum. The worst overall result was a fall to 56% for player 7-LL after $10,000$ games (not presented).

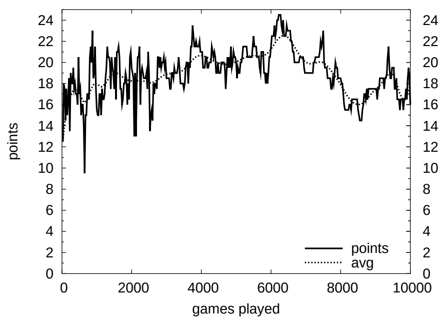
4.3 Learning on 25 Strong Opponents

After 10,000 games with 25 random opponents, the learning players were trained against 25 strong opponents during another 10,000 games. This time the opponents used the weight vectors of the 20 learning players trained using the LL and LB methods in the first 10,000 games. The additional 5 opponents were initialized with random weight vectors. Like before, the opponents did not change during training. This set of opponents was the same for all learning players in this phase. The results are presented in Table 1(b). This time the LL method was superior to LB one. The difference in performance was about 10 to 20 percent points in favor of LL during all the games played. The history of performance changes for the best LL and LB players is presented in Figs. 1(c) and 1(d), resp.

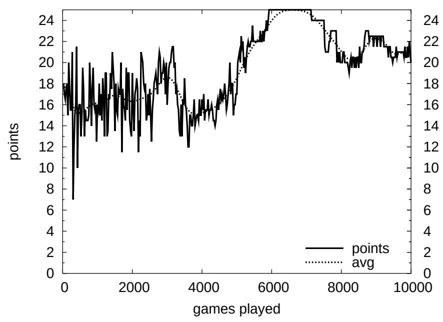
Results from the test phase are presented in Tables 1(c) and 1(d). They confirm the superiority of the LL method used for training with strong opponents. In this case however, unlike in the case of training with random opponents, the fall in training performance during games 7,500-10,000 was not observed. Further investigation of this phenomenon is one of our future research goals.



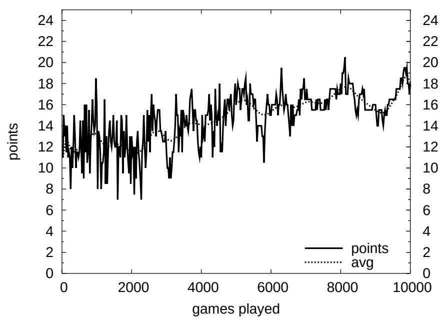
(a) Player 8-LL. Random opponents.



(b) Player 9-LB. Random opponents.



(c) Player 1-LL. Strong opponents.



(d) Player 10-LB. Strong opponents.

Fig. 1. History of performance changes for the best players in LL and LB methods playing against random and strong opponents.

5 Conclusions

A similar approach to the LL method presented in this paper was used earlier by Baxter [5]. In his chess learning program (KnightCap), there was no weight update in the case when the learning player won with a lower ranked opponent. As of our knowledge however, no one compared the LL and LB methods directly. The superiority of the LL method can be explained in the following way. A loss or a tie of the learning player means that the weight vector is not optimal and a weight update is recommended. A win however probably means only that the opponent was not good enough and a different one could have performed better. Learning on such outcomes can be misleading. The LL method showed its superiority only while learning on strong opponents. The reason behind this could be that while learning on random (weak) opponents no special care has to be taken in order to achieve good results and an ordinary TD(λ) algorithm is sufficient. Another explanation is that frequent weight updates may be desirable when playing against random opponents that share no common strategy.

Frequent weight changing in this case could compensate for the superiority of LL learning.

In the experiment also a variant of LL method (called 3L) consisted in training with the same opponent for up to 3 games in a row in case of losses was introduced. The idea of 3L method was to focus more on the opponents that "with no doubt" were stronger than the learning player. This method however appeared to be inferior to LL one.

A variant of the TD(λ) algorithm called TDLeaf(λ) was proposed in [5] for applications using a game tree search. Comparing the performance of this algorithm with the results presented in this paper is one of our future goals.

References

1. Sutton, R.: Learning to predict by the method of temporal differences. Machine Learning **3** (1988) 9–44
2. Tesauro, G.: Temporal difference learning and td-gammon. Communications of the ACM **38** (1995) 58–68
3. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3** (1959) 210–229
4. Schaeffer, J., Hlynka, M., Jussila, V.: Temporal difference learning applied to a high-performance game-playing program. In: International Joint Conference on Artificial Intelligence (IJCAI). (2001) 529–534
5. Baxter, J., Tridgell, A., Weaver, L.: Knightcap: A chess program that learns by combining td(lambda) with game-tree search. In: MACHINE LEARNING Proceedings of the Fifteenth International Conference (ICML '98), Madison WISCONSIN (1998) 28–36
6. Schraudolph, N.N., Dayan, P., Sejnowski, T.J.: Learning to evaluate go positions via temporal difference methods. In Baba, N., Jain, L., eds.: Computational Intelligence in Games. Volume 62. Springer Verlag, Berlin (2001)
7. Walker, S., Lister, R., Downs, T.: On self-learning patterns in the othello board game by the method of temporal differences. In: Proceedings of the 6th Australian Joint Conference on Artificial Intelligence, Melbourne, World Scientific (1993) 328–333
8. Alemanni, J.B.: Give-away checkers. http://perso.wanadoo.fr/alemanni/give_away.html (1993)
9. Schaeffer, J., Lake, R., Lu, P., Bryant, M.: Chinook: The world man-machine checkers champion. AI Magazine **17** (1996) 21–29
10. Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Machine Learning **22** (1996) 123–158