Comparison of TDLeaf(λ) and TD(λ) Learning in Game Playing Domain

Daniel Osman and Jacek Mańdziuk

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

Abstract. In this paper we compare the results of applying TD(λ) and TDLeaf(λ) algorithms to the game of give-away checkers. Experiments show comparable performance of both algorithms in general, although TDLeaf(λ) seems to be less vulnerable to weight over-fitting. Additional experiments were also performed in order to test three learning strategies used in self-play. The best performance was achieved when the weights were modified only after non-positive game outcomes, and also in the case when the training procedure was focused on stronger opponents. TD-learning results are also compared with a pseudo-evolutionary training method.

1 Introduction

The Temporal Difference TD(λ) algorithm [1] has been successfully used for learning optimal control in domains with a large state space. Some of the well known applications involving TD(λ) are TDGammon [2] (computer backgammon), KnightCap [3] (computer chess), TDL Chinook [4] (computer checkers) and computer Go [5].

In [6] we have applied the TD(λ) algorithm in the domain of give-away checkers (GAC). The game shares the same rules of playing [7] as regular checkers. The only difference is the goal. In GAC a player wins if no legal move can be made in his turn. The game at first glance may seem trivial or at least not interesting. However a closer look reveals that a strong positional knowledge is required in order to win. A simple piece disadvantage isn't a good estimation of in-game player's performance. Due to the fact that GAC is not a very popular game, we did not concentrate at this point, on creating a master GAC playing program. Our aim was to show that in a domain with a large state space, a control learning program can benefit from the Temporal Difference algorithm even when a relatively simple value function is used (with only 22 weights).

Continuing the work started in [6], we now extend the experiment by testing the TDLeaf(λ) algorithm [3]. TDLeaf(λ) is a modification of TD(λ), enhanced for use in domains, where a d-step look ahead state search is performed in order to choose an action to be executed at a given time step.

We also test a pseudo-evolutionary learning method (EVO) described in [8] and compare it with TD(λ) and TDLeaf(λ).

Several possible training strategies related to Temporal Difference learning are possible in practice. Results presented in [6] show that learning only on non-positive game outcomes i.e. (loss or tie) is much more efficient than learning on all games. In this paper (section 4) we propose and verify a new learning strategy denoted by L3 which consists in playing up to three games in a row against opponents that are stronger than the learning player. The results are very promising.

2 Value Function, TD(λ) and TDLeaf(λ) Algorithms

The value function is used to assign values to states. The value of state s is an approximation of the final game outcome accessible from s. The possible game outcomes are $+100$ for win, -100 for loss and 0 for tie. Each state is defined by a limited number of features (22 in our experiment). The list of features implemented was based on the one used by Samuel in [9]. During the game, the following value function was used:

$$V(s,w) = a \cdot \tanh\Big(b \cdot \sum_{k=1}^{K} \omega_k \cdot \phi_k(s)\Big), \qquad a = 99,\ b = 0.027,\ K = 22 \tag{1}$$

where $\phi_1(s), \ldots, \phi_K(s)$ are state features and $w = [\omega_1, \ldots, \omega_K]^T \in \mathbb{R}^K$ is the tunable weight vector. Parameter $a = 99$ to guarantee that $V(s,w) \in (-99; +99)$ and $b = 0.027$ in order to decrease the steepness of the $\tanh(\cdot)$ function.

The TD(λ) and TDLeaf(λ) algorithms are used in order to modify the weights of $V(s,w)$. The goal of this weight correction is to achieve a perfect value function, that is the one that always returns the correct game outcome prediction from any given state $s \in S$. In TDLeaf(λ) [3] the equation for modifying weights is as follows:

$$\Delta w = \alpha \cdot \sum_{t=1}^{N-1} \nabla_w V(s_t^{(l)}, w) \cdot \sum_{i=t}^{N-1} \lambda^{i-t} \cdot d_i \tag{2}$$

where $s_1, s_2, \ldots, s_N$ are the states observed by the learning player during the entire course of the game and $s_1^{(l)}, s_2^{(l)}, \ldots, s_N^{(l)}$ are the principal variation leaf nodes calculated for these states. Parameter $\alpha \in (0,1)$ is the learning step size and $\lambda \in (0,1)$ is the decay constant. $\nabla_w V(s_t^{(l)}, w)$ is the gradient of $V(s_t^{(l)}, w)$ relative to weights w and $d_i = V(s_{i+1}^{(l)}, w) - V(s_i^{(l)}, w)$ represents the temporal difference in state values obtained after a transition from state s_i to s_{i+1}.

The difference between TD(λ) and TDLeaf(λ) is that in TD(λ) the gradient at time step t in (2) is calculated for $V(s_t, w)$ as opposed to $V(s_t^{(l)}, w)$ in TDLeaf(λ).

3 Experiment Design

Ten learning players were trained in the experiment. Players 1 to 5 played using white pieces (they performed the initial move). Players 6 to 10 played using red

pieces. The look ahead search depth was set to $d = 4$. In the first stage, learning players 1 and 6 started with all weights set to zero. The rest of the learning players had their weights initialized with random numbers from interval $(-10, +10)$. Each learning player played in total $10,000$ games against a distinct set of 25 random opponents. The opponents in subsequent games were chosen according to some predefined permutation. All 250 opponents were pairwise different and had their weights initialized randomly from interval $(-10, +10)$.

In the second stage, each learning player started out with the weights that were obtained after finishing the first training stage. This time each learning player played $10,000$ games against a common set of 25 opponents. 20 of them were randomly picked from the learning players being developed in the first stage at different points in time. The remaining 5 opponents had their weights initialized randomly from interval $(-10, +10)$. The opponents were not modified during training.

Results obtained in the first stage were presented in [6]. Here, we report the results of the second stage of the training phase together with a test (validation) phase which involved matching up every learning player with 100 strong opponents developed in another experiment. These test opponents were not encountered earlier during training. One test phase match was repeated after every 250 training games. There was no weight modification during the test phase. The purpose of this phase was to test the general performance of the learning players. For every win the learning player was rewarded with 1 point, 0.5 for a tie and 0 points for a loss. The percentage results presented in the next section show the fraction of points received by the learning player out of the total number of points available.

In the pseudo-evolutionary learning method (EVO) only one opponent was used during training. This opponent was modified after every two games by adding Gaussian noise to all of its weights. If the learning player lost both games or lost and drawn then its weights were shifted by 5% in the direction of the opponent's weight vector. The two players changed sides after every game. We present the results of the EVO method obtained during training games 1-10,000 since in subsequent games (10,001-20,000) the performance started to gradually decrease.

4 TD(λ), TDLeaf(λ) and EVO Results

Choosing the best learning strategy for Temporal Difference method is still an open question. Many authors, for example [5] and [3], stressed the importance of this aspect. Besides tuning parameters α and λ in (2), it is even more important to properly choose the quality and the number of opponents that the learning player will play against. One must also choose whether the weights of the learning player are to change after each game (this approach will be denoted by LB) or only after games that the learning player lost or drawn (denoted by LL). Another question is whether to play the equal number of times against each opponent or to concentrate on the stronger ones (this strategy will be denoted

Table 1. TD, TDLeaf and EVO average results.

(a) Training phase results.

games	TDLeaf+LL	TDLeaf+LB	TD+LL	TD+LB	TD+L3	EVO
1 - 2,500	61.3%	59.0%	61.8%	52.9%	56.7%	64.5%
2,501 - 5,000	66.3%	65.6%	64.3%	54.9%	45.5%	63.5%
5,001 - 7,500	60.1%	63.3%	71.9%	51.8%	40.8%	64.8%
7,501 - 10,000	60.1%	64.6%	62.8%	52.3%	37.5%	63.1%

(b) Test phase results (against strong opponents).

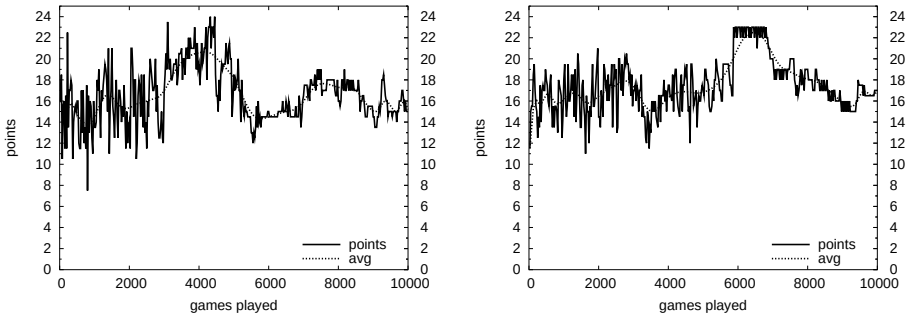
games	TDLeaf+LL	TDLeaf+LB	TD+LL	TD+LB	TD+L3	EVO
1 - 2,500	51.7%	50.4%	53.3%	43.3%	53.3%	30.1%
2,501 - 5,000	52.2%	55.0%	56.7%	44.8%	57.4%	32.9%
5,001 - 7,500	50.6%	56.1%	54.0%	47.9%	56.7%	35.3%
7,501 - 10,000	50.8%	56.0%	54.8%	48.2%	56.1%	36.8%

by L3). Strategy L3, proposed by the authors, is a modification of LL. In L3 the learning player after losing a game plays the next game with the same opponent (not more than three times in a row, however). This causes the learning process to be concentrated on stronger opponents.

The LB learning strategy for TDLeaf was enhanced (similarly to [3]) by preventing the learning player from learning on the opponents' blunders (more precisely: state transitions that were not predicted by the learning player). This limitation of the number of states that the learning player could learn on was applied only when the learning player won with the opponent. In case of a loss or tie, the learning player learned on all encountered states.

The results of the training phase for all learning strategies presented in Table 1(a) clearly show the superiority of the TD algorithm using the LL learning strategy (TD+LL). The worst performance was obtained by TD+L3, which can be explained in the following way: in L3 the learning player concentrates more on stronger opponents. Since in case of losing the game, the learning player was forced to play against the same opponent again, it turned out that the learning player often lost 3 games in a row when the opponent was really strong. Moreover, due to decrease of α in time the above situation (3 loses in a row) happened more frequently in subsequent training games than at the beginning of the training period. This observation was supported by the following results obtained when the learning coefficient α was kept constant: 56.7%, 45.5%, 44.5%, 44.0%.

In [3] the TDLeaf algorithm showed a faster performance increase than plain TD when learning to play turbo chess. In our experiments indeed TDLeaf achieved the maximum of its performance (66% in games 2,501-5000) earlier than TD (72% in games 5,001-7,500). This is also presented in Figs 1(a) and 1(b). There were a few TDLeaf learning players during games 2,501-5,000 that performed particularly well. However as can be seen in Table 1(a), the overall average training performance of TDLeaf was inferior to TD.



(a) Player 3-TDLeaf LL. (b) Player 3-TD LL.

Fig. 1. History of performance changes for the best learning players using TDLeaf+LL and TD+LL algorithms during the training phase.

Table 1(b) presents the results of the test phase. This time the results of TDLeaf+LB and TD+LL were comparable. It is important to note that for TDLeaf the LB learning strategy was superior to LL. The most superior results were achieved by TD+L3 method, which confirms the efficacy of L3 learning strategy. The raise in performance from 64.3% to 71.9% by TD+LL in the training phase was not reflected in the test phase where in the same time a fall from 56.7% to 54.0% was observed. This phenomenon of increasing training phase performance along with decreasing test phase results (over-fitting) was only sporadically observed when using the TDLeaf algorithm.

In our experiments the EVO method turned out to be significantly inferior to TD in the test phase (Table 1(b)). Training results for EVO should not be directly compared with the remaining ones in Table 1(a) because of different training strategy used. A similar learning method was used earlier in [8] where its performance was superior to TD. This could have been caused by different characteristics of the games chosen (GAC vs Rummy). Moreover, in our experiments a more sophisticated TD self-play learning strategy had been used enabling better state space exploration due to frequent opponent changing.

In another test phase (not presented in the paper) that consisted in playing against 100 randomly generated opponents the results reached the level of 70-75% for TD and TDLeaf and 60% for EVO. Although the results may not seem very high, they do show the amount of improvement obtained with the Temporal Difference algorithm and promise better performance if a more sophisticated value function were used.

5 Conclusions

The main conclusion from this work is that in the training phase, the TDLeaf algorithm shows overall inferior performance compared to TD. In the test phase however, the results of both algorithms are comparable. The TDLeaf algorithm

has an additional benefit of faster performance improvement and seems to be less vulnerable to weight over-fitting which results in good training phase results combined however with an average test phase performance.

The TD learning methods are visibly more efficient than the pseudo-evolutionary technique described in [8]. The results also confirm that self-play can be successfully used in Temporal Difference learning applied in a deterministic game of give-away checkers as long as frequent opponent changing is guaranteed. As it was mentioned in [3], to achieve a good level of play, one must match up the learning player against opponents with strengths similar to that of the learning player. Playing against many different opponents (25 in our case) ensures adequate state space exploration. There is a great chance that once we encounter the same opponent again, the learning player's weights will not be the same as before and therefore the game will take a different course. Secondly in case of many opponents the chance that some of them will share a similar strength of play as the learning player increases.

The L3 training strategy, in which the learning player is more focused on strong opponents was inferior in the training phase but achieved the best score in the test phase. Closer investigation of L3 method is one of our current research 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. Baxter, J., Tridgell, A., Weaver, L.: Knightcap: A chess program that learns by combining td(λ) with game-tree search. In: Machine Learning, Proceedings of the Fifteenth International Conference (ICML '98), Madison Wisconsin (1998) 28–36
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. 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)
6. Mańdziuk, J., Osman, D.: Temporal difference approach to playing give-away checkers. In Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A., eds.: 7th International Conference on Artificial Intelligence and Soft Computing (ICAISC 2004), 7th International Conference, Zakopane, Poland. Volume 3070 of Lecture Notes in Computer Science., Springer (2004) 909–914
7. Alemanni, J.B.: Give-away checkers. http://perso.wanadoo.fr/alemanni/give_away.html (1993)
8. Kotnik, C., Kalita, J.K.: The significance of temporal-difference learning in self-play training td-rummy versus evo-rummy. In Fawcett, T., Mishra, N., eds.: Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), Washington, DC, USA, AAAI Press (2003) 369–375
9. Samuel, A.L.: Some studies in machine learning using the game of checkers. IBM Journal of Research and Development **3** (1959) 210–229