Improving Temporal Difference Learning Performance in Backgammon Variants

Nikolaos Papahristou and Ioannis Refanidis

University of Macedonia, Department of Applied Informatics,
Egnatia 156, Thessaloniki, 54006, Greece
nikpapa@uom.gr, yrefanid@uom.gr

Abstract. *Palamedes* is an ongoing project for building expert playing bots that can play backgammon variants. As in all successful modern backgammon programs, it is based on neural networks trained using temporal difference learning. This paper improves upon the training method that we used in our previous approach for the two backgammon variants popular in Greece and neighboring countries, Plakoto and Fevga. We show that the proposed methods result both in faster learning as well as better performance. We also present insights about the selection of the features in our experiments that can be useful to temporal difference learning in other games as well.

1 Introduction

Backgammon is an ancient board game of luck and skill that is very popular throughout the world with numerous tournaments and many popular variants. Variants of any game usually aren't as interesting as the standard version, but often offer a break in the monotony of playing the same game over and over again. Backgammon variants come in different flavors: some change the standard backgammon rules only slightly (Portes), while others have different rules for moving the checkers (Fevga, Plakoto), alternate starting positions (Nackgammon), different checker direction (Fevga) or assign special value to certain dice rolls (Acey-Deucey, Gul-bara). *Palamedes* [6] (Fig. 1) is an ongoing project dedicated to offer expert-level playing programs for backgammon variants.

The objective for each player of virtually all variants is to move all his checkers to the last quadrant (called the *home board*), so he can start removing them; a process called *bearing off*. The player that removes all his checkers first is the winner of the game. Players may also win a double game (2 points) when no checker of the opponent has been beared-off. A triple win and the doubling cube are usually used only in standard backgammon.

In previous work [7], following the successful example of TD-Gammon [14,15,16] and other top playing backgammon programs, we trained neural networks (NN) using temporal difference learning for playing *Plakoto* and *Fevga,* two variants very popular in Greece and neighboring countries. The contribution of this paper is the improvement of our training methods and the presentation of new results for these games. More specifically, we improved the performance of the training method by making the target of the updates to be the inverted value of the next player's position;

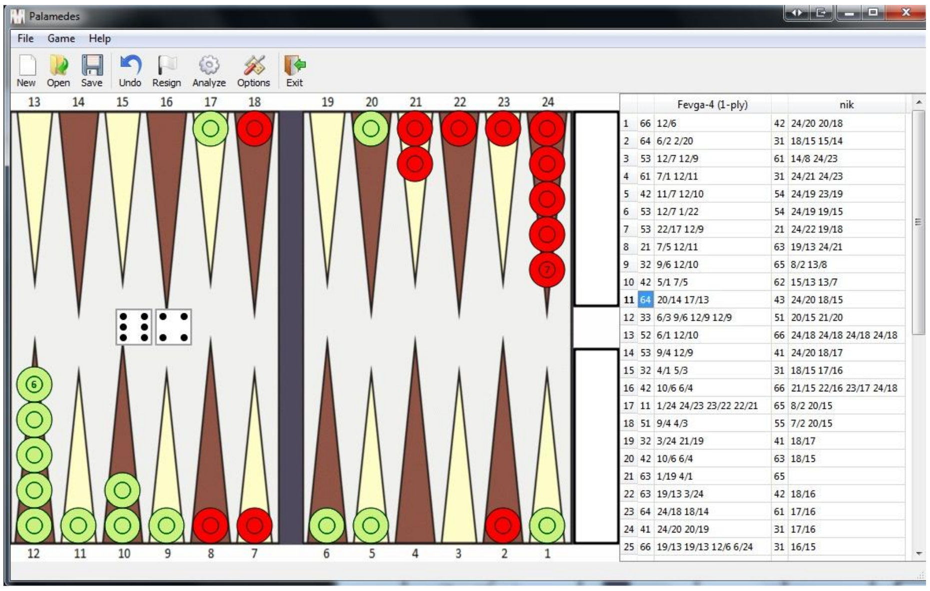
we also improved a little the learning speed by updating the positions starting from the end and recalculating the target value. Furthermore, we identified problems with the generalization of the neural network at certain positions of the Plakoto variant and we present our solution based on adjusting the input features along with analysis to the cause of the problem and its implications to learning from manually added features in general.

The remaining of the section describes the main rules of Plakoto and Fevga variants and compares the complexity of the games with standard backgammon. The complete set of rules for standard backgammon, Plakoto, Fevga and other variants can be found in [1].

1.1 Plakoto

The key feature of game Plakoto is the ability to pin hostile checkers, so as to prevent their movement. The general rules of the game are the same as the regular backgammon apart from the procedure of hitting. Players start the game with fifteen checkers placed in opposing corners and move around the board in opposite directions till they reach their home boards which are located opposite of the starting area.

When a checker of a player is alone in a point, the opponent can move a checker of his own in this point thus pinning (or trapping) the opponent's checker. This point counts then as a *made point* as in regular backgammon, which means that the pinning player can move checkers in this point while the pinned player cannot. The pinned checker is allowed to move normally only when all opponent pinning checkers have left the point (*unpinning*). Pinning the starting point of the opponent results immediately in a double win.



		Fevga-4 (1-ply)		nik
1	66	12/6	42	24/20 20/18
2	64	6/2 2/20	31	18/15 15/14
3	53	12/7 12/9	61	14/8 24/23
4	61	7/1 12/11	31	24/21 24/23
5	42	11/7 12/10	54	24/19 23/19
6	53	12/7 1/22	54	24/19 19/15
7	53	22/17 12/9	21	24/22 19/18
8	21	7/5 12/11	63	19/13 24/21
9	32	9/6 12/10	65	8/2 13/8
10	42	5/1 7/5	62	15/13 13/7
11	64	20/14 17/13	43	24/20 18/15
12	33	6/3 9/6 12/9 12/9	51	20/15 21/20
13	52	6/1 12/10	66	24/18 24/18 24/18 24/18
14	53	9/4 12/9	41	24/20 18/17
15	32	4/1 5/3	31	18/15 17/16
16	42	10/6 6/4	66	21/15 22/16 23/17 24/18
17	11	1/24 24/23 23/22 22/21	65	8/2 20/15
18	51	9/4 4/3	55	7/2 20/15
19	32	3/24 21/19	41	18/17
20	42	10/6 6/4	63	18/15
21	63	1/19 4/1	65	
22	63	19/13 3/24	42	18/16
23	64	24/18 18/14	61	17/16
24	41	24/20 20/19	31	17/16
25	66	19/13 19/13 12/6 6/24	31	16/15

Fig. 1. ***Palamedes*****:** A program for playing backgammon and variants

1.2 Fevga

The main difference of Fevga is that there is no pinning or hitting. If the player has even a single checker in one point, this point counts as a *made point*, effectively preventing the movement of the opponent's checkers in this point. Each player starts with fifteen checkers on the rightmost point of the far side of the board, at diagonally opposite corners from each other, whereas the two players move in the same direction. A crucial characteristic of this variant is that a prime formation (six consecutive made points) can be more easily achieved than in backgammon. Consequently, human experts usually plan their strategies always having in mind to create some kind of prime in order to block the opponent as soon as possible in his development, while at the same time prevent the opponent from blocking them.

1.3 Complexity compared to standard backgammon

Table 1 summarizes the differences in complexity of backgammon, Plakoto and Fevga variants. Standard backgammon state space has been computed in excess of 10^{20} states [16]. The Fevga variant without having a bar position (a position where the hitted checkers are placed in standard backgammon) inevitably has somewhat fewer states, whereas the possibility of pinning gives Plakoto a higher number of total states. The numbers shown for the average branching factor and the average game length in table 1 were computed by taking the best agent at our disposal for each game and making it play 50,000 games against itself. We used Plakoto-3 for Plakoto, Fevga-5 for Fevga and a NN trained with expert features for backgammon. This backgammon NN has a performance of 0.608 against the pubeval [8] benchmark program.

Table 1: State space size and game tree complexity

Game	State Space Size	Branching Factor (avg)	Game Length (avg)
Backgammon (BG)	$> 10^{20}$	16	55
Plakoto	> BG	23	92
Fevga	< BG	25	91

One common difference of both games compared to standard backgammon is that they last longer: whereas a standard backgammon game lasts on average 55 plies, a game of Fevga and Plakoto lasts on average around 92 plies. The game of Fevga is the most straightforward of the three. Players run their checkers to their home board resulting in game sequences with more or less same total ply count. In standard backgammon the possibility of hitting sometimes results in long sequences. In Plakoto the possibility of pinning the opponent's starting point can result in shorter than usual total plies in a game. On the other hand, when both players have pinned opponent checkers in their home board, the game lasts longer than usual (up to 150 plies) because of a large number of no-move plies.

Apart from longer sequences, Plakoto and Fevga variants have larger average branching factors. According to our experiments, the branching factor (in non-chance layers) of standard backgammon is 16[1], while for Fevga it is 25 and for Plakoto 23. This is mainly due to the fact that standard backgammon has fewer middle game positions where the number of available moves is at its peak.

2 Updating the temporal difference in games

The theoretical background of reinforcement learning [11,12] and the TD(λ) [10] temporal difference learning algorithm that we used is described in [7]. This section explores alternatives for selecting the target of the TD updates and for the creation and updating of a self-play game sequence.

2.1 Determining the target of the update

In order to find the best move in a given situation, backgammon programs usually score each possible afterstate (that is the states resulting *after* the player has played a move) and select the move that produces the afterstate with the biggest score.

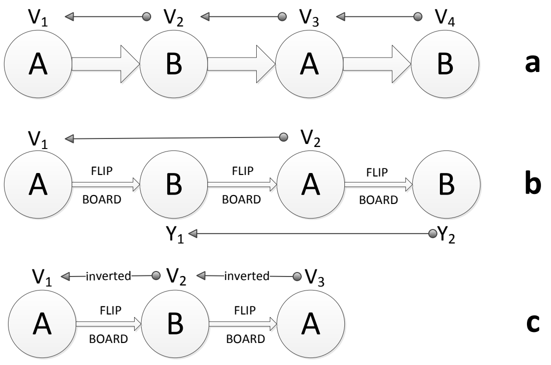


Fig. 2. Alternate updating methods of the temporal difference in two player zero-sum games. Method a: Update the values without flipping the board. Requires input(s) to designate which player is on the move. Method b: Updates are split in two. Method c: Updates are done on the inverted value of the next player. Circles indicate a position after a player (A or B) has made a move (afterstate).

An important implementation detail for a TD+NN learning system is the selection of input-target pairings for the TD update. In previous work we split up each training game into two training sequences, one for the afterstates of the first player and another for the afterstates of the second player, and we updated these sequences separately (Fig. 2.b). In this work we made one simple, yet very effective improvement: instead of splitting up each training game in two, we keep one training

[1] Several sources (eg [15]) claim a branching factor of 20 for standard backgammon. This number may have been calculated in conjunction with resignations and doubling cube drops.

sequence and we update each player's afterstate using as target the inverted value of the other's player afterstate on the next move (Fig. 2.c). Both methods flip the board so as both players' afterstates are given to the neural network as if it is the first player to move. This is different from the approach used originally by TD-Gammon (Fig. 2.a), where there was no flipping of the board and the neural network learned to play the game for both sides, identifying the side to move by two binary inputs. We believe the board-flipping approach has the potential of getting improved performance as the expressiveness of the neural network is increased.

2.2 Sequence creation and how to update

Contrary to standard backgammon, when we started making programs for the variants Plakoto and Fevga, there was no other program close to expert play, nor were databases of games available. Therefore, self-play was for us the only option for creating a game sequence. We examined the following options for creating and updating a self-play game:

1) Learning online (each update is done immediately after a move is played).
2) Learning offline (updates are done incrementally after the game ends)
   a) Forward offline: Updates are done starting from the first position of the game and ending at the terminal position.
   b) Reverse offline: Updates are done starting from the terminal position of the game and ending at the first.
   c) Reverse offline recalc: As previous, but recalculate target value after each update.

The intuition of updating backwards an offline game is that updates of non-terminal states will be more informed as the reward of the outcome of the game is received on the first update of the game. This is enhanced with the addition of the recalculation of the target value. Online updates have the benefit of learning while the game is in progress; however there is a chance that at the start of training, where moves are more or less random, the agent will get stuck or progress slowly.

Preliminary experiments with all of the above methods showed that the slowest method was forward offline, particularly in the Fevga variant, with the others resulting in more or less the same performance (Fig. 3). The reverse offline method with recalculation of the target value learns the fastest than all others at the start of the training and continues to have good performance afterwards. The downside is that more computation is needed in order to recalculate the target value at every step. However, this was not felt in our case since the creation of a game sequence is much more time consuming than the time to make the updates. Even with slower learning progress, all methods were found to reach the same level, so whatever final performance gains described later in the paper were only due to changing the updating method from (b) to (c) (2.1).

In our previous experiments, we used the forward offline method. Following the experiments mentioned in this section, all experiments in this paper were conducted using reverse offline with target recalculation.

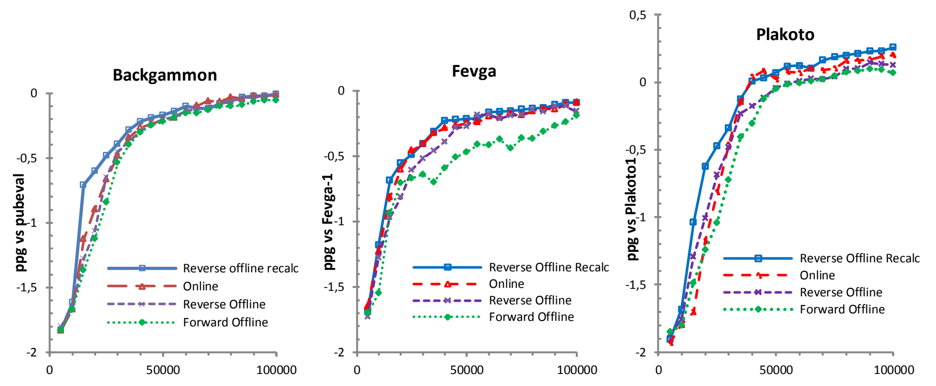


Fig. 3. Training progress of methods for sequence creation and update in Backgammon (left), Fevga (middle) and Plakoto (right). Every line is the average of 10 different training runs starting from the same random weights. For speed reasons, NNs in all games have 10 hidden units and no expert features. Benchmark opponents are pubeval for backgammon, Fevga-1 for Fevga, and Plakoto-1 for Plakoto.

3 Experimental Results

We compared the proposed training method to the previous one [7] by training again the NNs with the added expert features. For Plakoto, the new agent was named Plakoto3 and has exactly the same inputs as Plakoto2. For Fevga, we trained two new NNs: Fevga-4 has the same procedure as Fevga-2, whereas Fevga-5 has the same intermediate reward as Fevga-3. Table 2 shows all the techniques used by the various versions examined in this paper.

3.1 Results in the Plakoto and Fevga variants

Earlier results [7] showed that Fevga-2's strategy was much different from the one considered by the human experts, even with features recognized the presence of primes (six consecutive made points) in a position. To clarify the importance of primes more precisely, a new NN was trained (Fevga-3) where the agent learned with the same input units as Fevga-2, but with one important difference: when reaching a position with a prime formation, the target of the TD update was made a constant value instead of the next position value. This had the result that the learned strategy was based on the creation of primes, which is roughly equivalent to what is perceived by experts as the best strategy. Results showed that the riskier strategy of Fevga-2 scores more points against the benchmark program Tavli3D[13] than Fevga-3, but when getting them to play against each other Fevga-2 was a little bit inferior. To preserve continuity with our previous work, we continued to benchmark our training progress with the open source program Tavli3D, which at the time of writing was the only open source program that can play these variants.

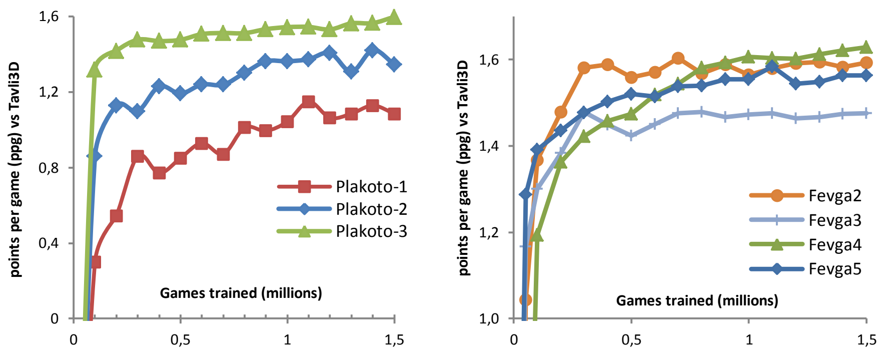


Fig. 4. Training progress of all trained NNs against the Tavli3D benchmark program in the Plakoto variant (**Left**) and the Fevga variant (**Right**).

All networks had 100 hidden neurons and were trained to 1.5 million games. For simplicity, we fixed the value of λ to zero for the experiments conducted in this paper. For λ>0 and reverse updates, care must be taken when taking future time steps into consideration: since every time step is viewed as the first player, any value taken by future time steps that is not a move by the player making the current update must be inverted. As the initial training of Fevga-2 and Fevga-3 were only 700,000 games, we extended their training (with the same initial λ=0.7) to match the new ones. During the training, we periodically saved the weights of each NN and we tested the networks against Tavli3D for 10,000 test games each, half as the first player and half as the second player (Fig.4). The result of the tested games sum up to the form of estimated *points per game* (ppg) and is calculated as the mean of the points won and lost.

We also tested the best set of weights of each NN by playing tournaments against each other at (1-ply) as well as by implementing a simple look-ahead procedure using the expectimax algorithm [5] at 2-ply depth (Table 3). In order to speed up the testing time, this expansion of depth-2 was performed only for the best 15 candidate moves (forward pruning). For the same reason, the total amount of testing games using 2-ply was reduced to 1,000 per test.

Table 2: Summary of techniques used by the various agents

Plakoto Agent	**Updating method (Fig. 2)**	**Sequence creation and update direction**	**Fevga agent**	**Updating method (Fig. 2)**	**Sequence creation and update direction**	**Intermediate reward**
Plakoto-1	b	Forward offline	**Fevga-2**	b	Forward offline	No
Plakoto-2	b	Forward offline	**Fevga-3**	b	Forward offline	Yes
Plakoto-3	c	Reverse offline recalc	**Fevga-4**	c	Reverse offline recalc	No
			Fevga-5	c	Reverse offline recalc	Yes

Table 3: Comparison of various agents at 1-ply and 2-ply for Plakoto (Left) and Fevga (Right). All results are in points per game (ppg) with respect to the player on the row. Players on columns always use 1-ply.

	Tavli3D	**Plakoto1**	**Plakoto2**		**Tavli3D**	**Fevga-2**	**Fevga-3**	**Fevga-4**
Plakoto-1	1-ply: +1.15 2-ply: +1.36	*	*	**Fevga-2**	1-ply: +1.60 2-ply: +1.61	*	*	*
Plakoto-2	1-ply: +1.46 2-ply: +1.60	1-ply: +0.98 2-ply: +1.35	*	**Fevga-3**	1-ply: +1.52 2-ply: +1.53	1-ply: +0.03 2-ply: +0.49	*	*
Plakoto-3	1-ply: +1.60 2-ply: +1.68	1-ply: +1.10 2-ply: +1.24	1-ply: +0.35 2-ply: +0.62	**Fevga-4**	1-ply: +1.63 2-ply: +1.64	1-ply: +0.35 2-ply: +0.53	1-ply: +0.26 2-ply: +0.48	*
				Fevga-5	1-ply: +1.58 2-ply: +1.59	1-ply: +0.42 2-ply: +0.60	1-ply: +0.32 2-ply: +0.45	1-ply: +0.02 2-ply: +0.14

Table 4. Analysis of some of the matches of Fevga-4 and Fevga-5

Match:	**Fevga-5 vs Fevga-4**		**Fevga-4 vs Tavli3D**		**Fevga-5 vs Tavli3D**	
	Fevga-5	**Fevga-4**	**Fevga-4**	**Tavli3D**	**Fevga-5**	**Tavli3D**
Single Wins	47.54%	39.52%	28.93%	2.74%	32.84%	2.86%
Double Wins	4.9%	8.04%	68.32%	0.01%	64.26%	0.04%
Total Wins	52.44%	47.56%	97.25%	2.75%	97%	3%
Final Score	+0.02ppg	-0.02ppg	+1.63ppg	-1.63ppg	+1.54ppg	-1.54ppg

The results in Plakoto show a significant increase in final performance. The performance of Plakoto-3 at 1-ply is equivalent to the performance of Plakoto-2 at 2-ply against Tavli3D. Additionally, Plakoto-3 learns faster than the other two agents.

In Fevga, Fevga-4 outperforms both Fevga-2 and Fevga-3 agents, while Fevga-5 outperforms all others except in the Tavli3D benchmark where it is inferior from Fevga-4, and Fevga-2 (Table 4). The explanation of this phenomenon is shown at Table 4: Against an inferior opponent Fevga-4 achieves more points because it wins more doubles due to its riskier strategy, while Fevga-5's safer strategy of building primes wins the same amount of games overall but fewer doubles. These new results show that the strategy learned by Fevga-4 and Fevga-5 is no different than the one learned from their previous counterparts (Fevga-2, and Fevga-3); simply the proposed training method learns the strategies better.

3.2 Feature selection

An interesting observation was made while testing the strategies that were learned in the Plakoto variant. The resulting strategy was very conservative with regard to its starting point (also called the "mother" point). The agent correctly identified that it must not let the starting point with one checker, as it would potentially be open to a pinning attack that would automatically lose the maximum amount of points (double game). However, it could not discriminate the positions that such an attack could not be carried out by the opponent, and protected its first point even after we added the expert feature of pinning probability in Plakoto2 and Plakoto3 (Fig. 5). This resulted in obvious errors in a small number of positions. For example, the amount of equity lost for selecting the wrong move in Fig. 5 was calculated to 0.276ppg by making a 100,000 games rollout on each of the moves in question (Table 5).

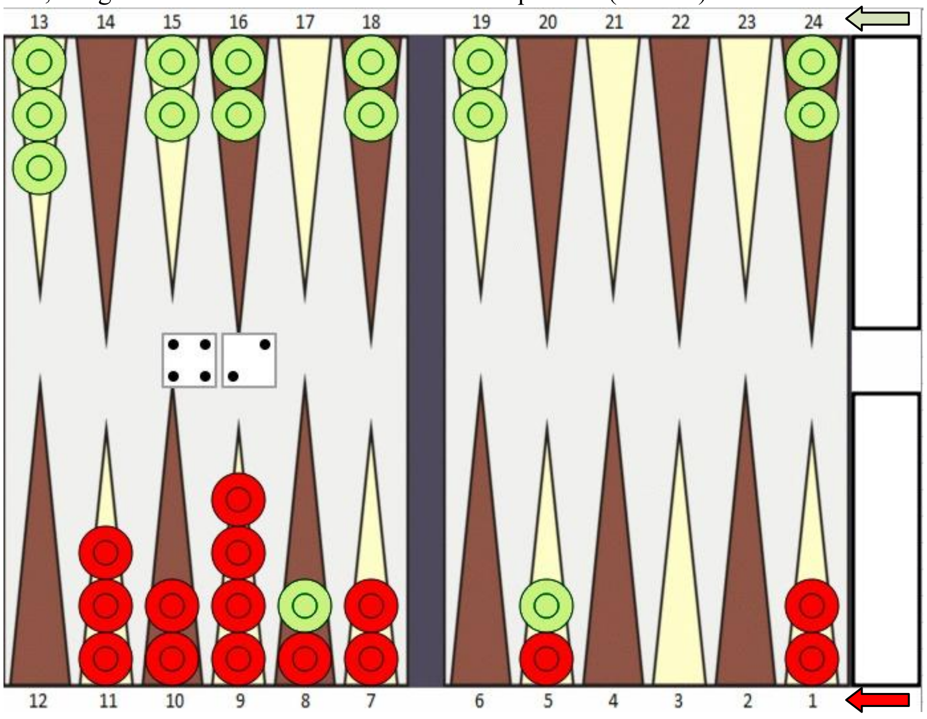


Fig. 5. Example of a position where agents Plakoto1-3 fail to produce the best move. The green player is to play roll 42. The best move here is 24/18 since the 24-point cannot be pinned by any dice roll. However, Plakoto1-3 agents prefer the clearly inferior move 24/20, 24/22 which give the opponent a pinning opportunity to get back into the game.

We suspected that the agent learned the harmful concept of leaving the first point open by the four raw features instead of the added expert feature "pinning probability at point 1". In order to confirm this we trained another agent (Plakoto-4) without the first of the four features for point 1, leaving only three features, one if 2 or more checkers are present, another if 3 or more checkers are present and a last one if 4 or more checkers are present. The resulting agent confirmed our suspicions, as it managed to learn the concept of "leaving the first point unprotected is bad" in a correct way, without committing the same mistakes of its predecessors. Evaluating Plakoto-4 final performance of 1.5 million trained games against Plakoto-3 in a 10,000 tournament resulted in equal performance. This may mean either a) positions of this kind do not appear frequently and when they appear they did not seem to have a significant impact to the result, or b) Plakoto-4 simply needs more training for the difference to tell.

Why was this concept not learned correctly by the other agents, especially when the other points where learned correctly? The concept of protecting the 1st point is one of the first things the agents learn, because it is the closest to the terminal position, the only position that receives reward, and because the random character of the first self-play training games result in many "mother doubles". When confronted with two features to learn the concept, one being a binary input, and one a float input between 0 and 1, the neural network chooses the first one because it is the easier and the faster to learn. It would appear that the agent would have a chance to “unlearn” this later as learning progresses when the estimates of the NN are closer to the optimal. However, this is never done as these kind of positions appear rarely, because the agent has learned how (wrongly) to defend against.

Table 5: Evaluation and rollout analysis of the two best moves of the position in Fig.5. The first four columns show the evaluation of the Plakoto-3 and Plakoto-4 NNs after 1-ply and 2-ply look-ahead. The fifth and sixth column show the equity of the position by making a rollout analysis using Plakoto-3 and Plakoto-4. The last column shows the equity that was lost by selecting the inferior move. The equity loss was calculated on the average of the two rollouts.

Move	Plakoto-3 (1-ply) eval	Plakoto-3 (2-ply) eval	Plakoto-4 (1-ply) eval	Plakoto-4 (2-ply) eval	Rollout Plakoto-3	Rollout Plakoto-4	equity loss
24/22 24/20	1.020	1.048	0.942	1.046	0.983	0.968	0.276
24/18	0.692	1.082	1.140	1.248	1.259	1.243	-

4 Related Work

Temporal difference learning has been used for learning an evaluation function in most modern games. The Knightcap program [2] learned an evaluation function for chess using TD-Leaf, an extension to TD(λ) where updates are made not on the resulting positions of a training game, but on the leaf nodes of the principal variations resulting from alpha-beta searches from the previous and next positions. However their approach only worked with initialization of the weights to a good starting point and could not learn from self-play. The rootstrap and treestrap algorithms introduced in [17] improved this approach by updating all interior nodes from the search tree towards the root node. Their program Meep managed to achieve a rating of 2,338 Elo on Internet Chess Club for blitz games.

TD-Leaf was also tried in backgammon [3], but the authors could not improve the performance compared to an already trained NN with TD(λ). Following these results, we didn’t perform any experiments with TD-Leaf since the training time will have increased significantly.

In checkers [9], TD-Leaf was able to tune the weights of the best program of all time Chinook, to the same level of a set of weights previously manually tuned for a period of 5 years. In their approach, two separate set weights were trained, one for white and one for black. The authors noted a similar performance for the two sets of weights, which indicates that a single set for both sides, as it was done with our approach, could be used.

In [18], a similar TD update like our proposed method was utilized for constructing a large architecture of several neural networks and for examining training using self-play and using an expert in the game of backgammon. In this work the update is called “minimax” update and the learning was conducted using the side of the first player only. Results show that learning when playing against an expert is faster at first but ultimately reaches the same performance as self-play. This is explained by the analysis in section 2.2. Starting with expert training and continuing with self-play is usually a good hybrid approach.

5 Conclusion and future work

We have managed to increase the performance of our temporal difference learning architecture in the backgammon variants Plakoto and Fevga by making the target of the update the inverted value of the opponent's next state and by updating the game sequence starting from the terminal and working to the starting position. The problems found by learning overlapping features indicate that one must choose the features to be trained very carefully, or else risking suboptimal performance. An automatic process of selecting, comparing and training the available features could be used in order to detect the beneficial from the problematic ones. This process, however, can be very time consuming, especially when many games must be played for good learning (as is in backgammon) or the number of features is large (as is in chess for example). These enhancements can be used in other games as well as in conjunction with other TD learning algorithms. As all our experiments were done with $\lambda=0$, an obvious continuation of this research is to determine if different values $\lambda>0$ can lead to improved performance.

We also plan to increase the number of backgammon variants that can be handled by *Palamedes*. Interesting candidates towards this direction are the acey-deucey, gioul and gul-bara variants. Our look-ahead procedure can be improved by searching in greater depths and by utilizing cutoff algorithms as in [4]. We are also planning to test *Palamedes* by participating in computer and human competitions.

Acknowledgements

This work was partially supported by the Greek State Scholarship Foundation (IKY). The authors would like to thank Anastasios Alexiadis for lending computing power for conducting some experiments presented in the paper, and the anonymous reviewers for their useful comments and suggestions.

References

1. BackGammon Variants, http://www.bkgm.com/variants
2. Baxter, J., Tridgell, A., Weaver, L., Knightcap: a chess program that learns by combining td(lambda) with game-tree search. In: Jude W. Shavlik (Ed.) *Proc. 15th International Conf. on Machine Learning* pp. 28–36. Morgan Kaufmann , San Francisco. CA. (2001)
3. Baxter, J., Tridgell, A., Weaver, L., Tdleaf(): Combining temporal difference learning with game-tree search. Australian Journal of Intelligent Information Processing Systems, 5(1), 39-43, (1998)
4. Hauk, T., Buro, M., Schaeffer , J.: *-minimax performance in backgammon. In: van den Herik, H., Bjornsson, Y., Netanyahu, N. (Eds.) Computers and Games 2006. LNCS, vol 3846, pp. 51-66 (2006)
5. Michie, D.. Game-playing and game-learning automata, In: L. Fox (Eds.) Advances in Programming and Non-Numerical Computation, pp 183-200., (1966)

6. *Palamedes*, http://csse.uom.gr/~nikpapa/software.html
7. Papahristou, N., Refanidis, I., Training Neural Networks to Play Backgammon Variants Using Reinforcement Learning, In: Cecilia Di Chio et al. (Eds.) EvoApplications 2011. LNCS, vol 6624, pp. 113-122, (2011)
8. Pubeval source code backgammon benchmark player, http://www.bkgm.com/rgb/rgb.cgi?view+610
9. Schaeffer, J., Hlynka, M., Vili, J.: Temporal Difference Learning Applied to a High-Performance Game-Playing Program. Proceedings IJCAI, pp. 529-534 (2001)
10. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning, 3(1), 9-44 (1988)
11. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Indroduction. MIT Press (1998)
12. Szepesvári, C.: Algorithms for Reinforcement Learning (Electronic Draft Version). (August 2010), http://www.sztaki.hu/~szcsaba/papers/RLAlgsInMDPs-lecture.pdf
13. Tavli3D, http://sourceforge.net/projects/tavli3d
14. Tesauro, G.: Practical issues in temporal differnce learning. Machine Learning, 4, 257-277, (1992)
15. Tesauro, G.: Programming backgammon using self-teching neural nets. Artificial Intelligence, 134, 181-199, (2002)
16. Tesauro, G. : Temporal Difference Learning and TD-Gammon. Communications of the ACM, 38(3), 58-68 (1995)
17. Veness, J., Silver, D., Uther, W., Blair, A.: Bootstrapping from Game Tree Search. Advances in Neural Information Processing Systems, vol 22, pp. 1937-1945, (2009)
18. Wiering, M.A.: Self-Play and Using an Expert to Learn to Play Backgammon with Temporal Difference Learning. Journal of Intelligent Learning Systems and Applications, 2, 57-68, (2010)