Training Neural Networks to Play Backgammon Variants Using Reinforcement Learning

Nikolaos Papahristou and Ioannis Refanidis

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

Abstract. Backgammon is a board game that has been studied considerably by computer scientists. Apart from standard backgammon, several yet unexplored variants of the game exist, which use the same board, number of checkers, and dice but may have different rules for moving the checkers, starting positions and movement direction. This paper studies two popular variants in Greece and neighboring countries, named Fevga and Plakoto. Using reinforcement learning and Neural Network function approximation we train agents that learn a game position evaluation function for these games. We show that the resulting agents significantly outperform the open-source program Tavli3D.

1 Introduction

Traditionally, the main way of developing software agents that play board games is to create an evaluation function that returns an estimate of the value of any specific position. This function is used to rank the available moves based on the resulting positions in order to choose the best one of them. The most common way to construct an evaluation function is to exploit a set of hand-designed features that identify important structures in a game position. Consequently, the output of the evaluation function is typically a weighted sum of all these features. The evaluation function can then be used with game-tree search algorithms to obtain better results.

The time needed for weight-tuning can be greatly reduced by using machine learning algorithms like Reinforcement Learning, which can learn through experience obtained either from already available games or by playing against an expert or through self-play. One of the most successful applications of reinforcement learning to game playing was the TD-Gammon program of Tesauro [11]. Using the temporal difference method of reinforcement learning, multilayer neural network function approximation, and self-play, TD-Gammon was able to learn to play extremely well even with little knowledge of the game. When knowledge-based features of the game and minimax search were added, the program was able to play at a level equal to the best human players [10]. Reinforcement learning has also been applied successfully to other board games [2,4,12]. Backgammon is an ancient board game of luck and skill that is very popular throughout the world with numerous tournaments and many popular variants. The game is conducted in a board containing 24 *points* divided in 4 quadrants of 6 points each. Each player starts the game with a number of checkers or stones in his disposal (usually 15) placed in fixed starting positions. The players take

C. Di Chio et al. (Eds.): EvoApplications 2011, Part I, LNCS 6624, pp. 113–122, 2011.

turns playing their checkers using an element of chance in the form of two six-sided dice according to the game rules. When all the checkers of a player are inside his last quadrant of the board (called the *home board*), he can start removing them; this is called *bearing off*. The player that removes all his checkers first is the winner of the game.

Apart from the standard backgammon game, many variants exist [1]. Some change the standard backgammon rules only slightly, while others have different rules for moving the checkers, alternate starting positions, different checker direction or assign special value to certain dice rolls. In this paper we examine two backgammon variants very popular in Greece and neighboring countries, called *Plakoto* and *Fevga*, which have very different rules than the regular backgammon game. As a result, the strategies used from expert human players are much different than regular backgammon. To the extent of our knowledge, these backgammon variants have not been studied before in the computer science literature.

We use the temporal difference learning method TD(λ) and self-play to train multilayer neural networks as evaluation functions of these game variants. Temporal difference learning uses the difference between the evaluations of two successive positions to update the evaluation of the first position. The goal of this study is to produce strong players for both the backgammon variants examined.[1]

2 Background

Reinforcement learning (RL) algorithms attempt to find a policy that maps states of the domain examined to the actions ought to take in those states [6,7]. In board games, states are equivalent to the game positions and actions are equivalent to player moves.

The environment is usually formulated as a finite-state Markov decision process (MDP) typically consisting of 1) a set of environment states, X (x_1, x_2, … x_T), 2) a set of actions A available to the agent, with the subset of actions applicable to state x denoted as $A(x)$, 3) a transition function, $P(x, a, y)$, which gives the probability of moving from state x to some other state y provided that action a was chosen is state x, 4) a reward function, $R(x, a, y)$, which gives the expected immediate reward when action a is chosen in state x resulting in state y.

At each time t, the agent being in $x_t \in X$ chooses an action $a \in A(x_t)$, perceives the new state x_{t+1} and receives the reward $r_t=R(x_t, a, x_{t+1})$. Based on these interactions the goal of the agent is to choose a behavior that maximizes the expected return.

2.1 TD(λ) Algorithm

There are numerous RL algorithms, one of the first and most simplest of which is Temporal Difference learning or TD(0) [5]. In TD(0) a value function $V(x)$ is estimated based on the expected value and return of the next state, performing the following calculations at every step t:

[1] A graphical user interface of the agents presented in this paper can be downloaded from `http://csse.uom.gr/~nikpapa/TavliGUI`

$$\delta_t = r_{t+1} + \gamma V(x_{t+1}) - V(x_t) \quad (1)$$

$$V(x_t) \leftarrow V(x_t) + a\delta_t \quad (2)$$

Where $\alpha \in [0,1]$ is a parameter called the learning rate. The algorithm updates the value of the state x_t towards the "target" value $r_{t+1} + \gamma V(x_{t+1})$.

TD(0) uses one step backups because it is based on the reward of one action (the next). It is possible to base the backup on more than one future rewards as some mixture of the multi-step return predictions. This results in the TD(λ) [5] algorithm which uses a factor $\lambda \in [0,1]$ to discount TD-errors of future time-steps:

$$V(x_t) \leftarrow V(x_t) + a \sum_{k=0} (\gamma\lambda)^k \delta_{t+k} \quad (3)$$

The λ factor determines how much the updates are influenced by events that occur later in time. For λ=0 the agent considers events of the next time step only, making the algorithm equivalent to TD(0). For λ=1 the algorithm considers only the terminal time step, resembling the procedure of Monte Carlo sampling. Intermediate values offer a way of determining how further in the future we go in order to update the present.

2.2 Neural Network Function Approximation

When the number of the states is very large, as in most real-world applications, computing and storing the values of all states is impractical. Backgammon state-space (calculated in excess of 10^{20} states) is an example of such a domain. Therefore, we often seek a way of generalizing from a limited subset of the states to the much larger complete state-space. This generalization is also known as function approximation.

Artificial neural networks are one of many function approximation methods used with RL that has been proven a successful choice for several applications. With this approach the weights of the neural network are used as function approximation to the value function and the backpropagation procedure is used to make the TD update according to the following equation for every output unit:

$$\Delta w_t = \alpha(Y_{t+1} - Y_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w Y_k \quad (4)$$

where w is the vector of neural network weights being tuned, Y_t is the prediction for the output at time step t, $\nabla_w Y_k$ is a set of partial derivatives for each component of the weights w, α is the standard learning rate, and λ is the factor controlling how much future estimates affect the current update.

3 The Games Plakoto and Fevga

In Greece there are three popular backgammon variants: *Portes*, *Plakoto* and *Fevga*. Portes is essentially the same game as the regular backgammon with the exception that there is no triple game win/loss (otherwise called backgammon). Since this game is virtually the same as regular backgammon, we targeted our research in the remaining two variants that have not been previously studied.

The games Plakoto and Fevga are played with different names but same rules in other countries in the Balkans, Russia and the Middle East [1]. Fevga is known as 'Narde' in Russia, Iran, Armenia, Azerbaijan, Georgia, Uzbekistan and as 'Moultezim' in Turkey. Plakoto is also known as 'Tapa' in Bulgaria and 'Mahbusa' in some Arab countries.

3.1 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 the home board which is located opposite from the starting area (Fig.1. left).

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 from the point (*unpinning*).

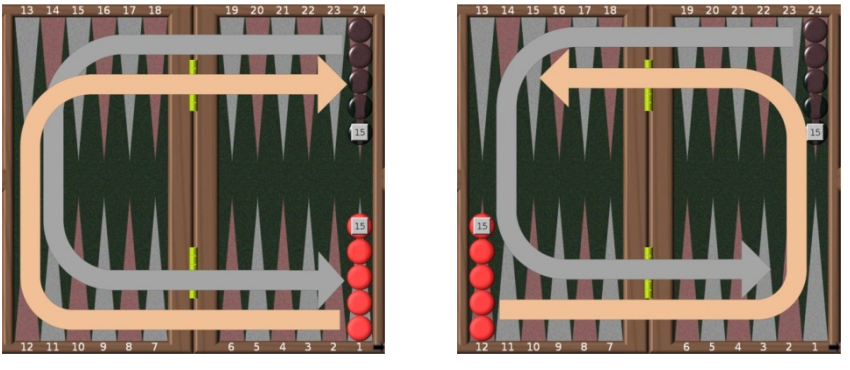

Fig. 1. Starting position and direction of play in the variants Plakoto(left) and Fevga(right)

3.2 Fevga

The main difference of Fevga from the other two games 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 (Fig.1. Right).

The game begins with a starting phase, where the players must move only one checker until it passes the opponent's starting point, before they may move any other of their checkers. The formation of *primes* (six consecutive made points) is easier in this game because a made point can be constructed with a single checker. The formation of primes has the following exceptions:

1. No player can form a prime in his starting quadrant.
2. No player can form a prime immediately in front of the opponent's starting point.

Finally, it is not permitted to completely block the opponent (*no-blocking rule*). This means that for a move to be allowed there must exist a dice roll that the opponent can use to move at least one checker.

4 Implementation

4.1 Learning Architecture

The architecture of the learning system that we used for all our experiments is shown in Fig.2

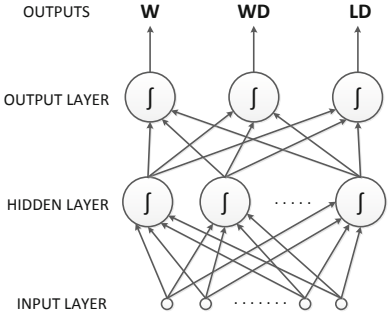

Fig. 2. The neural network architecture used in our learning system. All units of the hidden and output layer use sigmoid transfer functions.

The learning procedure is executed as follows: we start by creating a sequence of game positions beginning with the starting position and ending in the last position when the game is over. For each of these positions we use the backpropagation procedure of the neural network to compute the TD(λ) update, as described in Section 2. The self-play learning process is repeated until we can no longer improve the NN.

We used a modified version of the unary truncated encoding scheme used in [10] to map the board positioning of the checkers to the inputs of the neural network. We used three binary outputs to describe the final outcome of the game from the side of the first player. The first output (W) represents the outcome of the game, win or loss; the second output (WD) represents whether a double game is won; and the third output (LD) represents whether a double game is lost.

Under these training conditions, the neural network learns the "probability" of all three outputs at any time in the game, also called position equity. Whenever a move selection must be made, the agent scores the states resulting from all legal moves by combining all three outputs. For the creation of the game sequences, we used the

same neural network to select the moves for both sides. At every time-step the agent scores all legal moves available and selects the one with the highest score.

For the evaluation of the learned agents, two procedures were examined:

a) Evaluation against an independent benchmark opponent, the open source program Tavli3D 0.3.4.1 beta [8].

b) Evaluation against stored weights taken by the agent at different stages of the learning process. Examples are weights after 10^4 training games, weights after 10^5 training games etc.

During the training procedure the weights of the network were periodically saved and tested with procedures(a) and (b), until no more improvement was observed. All the tests were conducted in matches of 5000 games each. 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.

4.2 Determining the Learning Parameters

We conducted several experiments for each variant in order to find the best values of the various learning parameters. The final values of the most significant parameters are shown in table 1.

Table 1. Parameters Selected For Each Variant

Parameters	Fevga	Plakoto
Learning rate α	0.1	0.1
λ	0.7	0
Hidden Neurons	100	100

5 Empirical Evaluation

We used the same approach for both of the variants examined: First, we trained a neural network with inputs consisting only the raw position of the board. As with TD-Gammon, we observed a significant amount of learning even without the addition of "smart" features. It only took few thousands learning games for the agent to surpass the playing performance of the Tavli3D benchmark program. We evaluated the resulting agent and tried to improve its evaluation function by identifying expert features. A second neural network was trained from scratch including these expert features to the inputs of the neural network architecture.

5.1 Experiments in Fevga

The raw board position in the game of Fevga was encoded as follows: for every point of the board four binary inputs were used, each one designating whether there was one, two, three, or four and more checkers in the point. This coding thus used 96 input units for every player to encode the checkers inside the board and additional 2 units to encode the number of checkers off the board, for a total of 194 units. We named the agent trained with this coding scheme and the procedure described earlier **Fevga-1.**

Fevga-1 was assessed as average in strength by human standards. Concepts learned include the understanding of protecting the starting quadrant and attacking the starting quadrant of the opponent in the early phase of the game, as well as the smooth spreading of checkers. However, a major weakness was also found: a complete disregard for primes. The creation and sustainment of the prime formation is considered by human experts the most powerful strategy available in the Fevga variant.

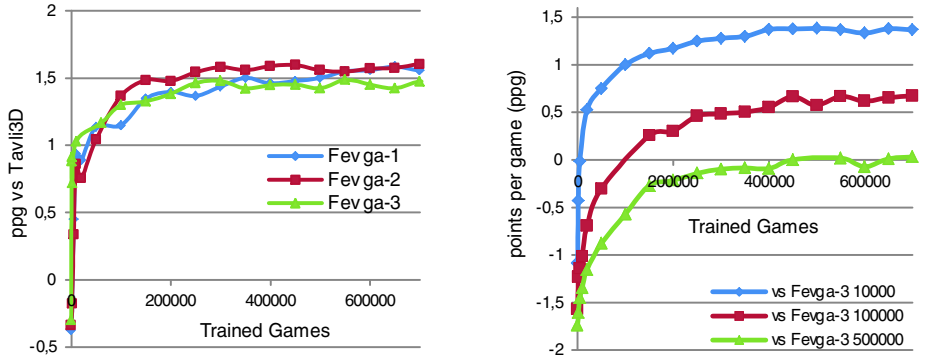

Fig. 3. Left. Training progress of all agents against the Tavli3D benchmark program. **Right.** Training progress of Fevga-3 against stored weights.

Adding Expert Features. Given the drawback in the playing strategy of Fevga-1, it was decided to add the special knowledge of primes in the inputs of the neural network as smart (or expert) features. The different formations of primes were divided in two categories according to their significance: a) early primes that are formed in the first two quadrants of the player and b) late primes that are formed in the last two quadrants as well as between the 4th and the 1st quadrant. Late primes are more powerful because they restrict the opponent earlier in his/her development and frequently result in the winning of a double game. These features take the form of four binary input units of the neural network that are enabled when the player and/or the opponent makes a prime and at least one opponent checker is left behind it. In addition, two more special features common to regular backgammon were also added: a) one input unit for the *pipcount* of each player, which is the total amount of points (or *pips*) that a player must move his checkers to bring them to the home board and bear them off, and b) two input units for the existence of a *race* situation, which is a situation in the game where the opposing forces have disengaged so there is no opportunity of further blocking. The total number of input units in this encoding (which we named Fevga-2) is 201.The evaluation of Fevga-2 (Fig.3.Left) showed only a marginal increase in performance that was verified by manual human analysis: while not totally ignorant for the value of prime formation as Fevga-1, Fevga-2 failed to grasp the essence of primes.

Adding Intermediate Reward. To clarify more precisely the importance of primes, a third neural network was trained 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 constant value was for primes of type (a) equivalent with winning a single game and for primes of type (b) equivalent with winning a double game. In other words, intermediate rewards were introduced when primes were formed in the game. This had the result that the strategy learned was a strategy based in the creation of primes, which is roughly equivalent to what is perceived by experts as the best strategy. We named this agent Fevga-3. Its training progress can be seen in Fig.3.Right.

Indeed after manual examination, the playing style of Fevga-3 was very similar to the way humans play the game. Not only did it recognize the value of primes and didn't lose opportunities to make one, but was also able to play moves that facilitated the creation of primes at later stages. The results of the evaluation against the Tavli3D benchmark show that Fevga-2 gains slightly more points per game than Fevga-3 (+1.61ppg vs +1.52ppg). However, when we compared Fevga-2 to Fevga-3 by selecting the best set of weights and playing 5000 games against each other, the results of this match showed a marginal superiority of the Fevga-3 player (+0.03ppg).

Table 2. Analysis of the match Fevga-2 vs Fevga-3

Result/Points	Fevga-2	Fevga-3	Total
Single Wins	1704 (34.08%)	2513 (50.26%)	4217 (84.34%)
Double Wins	556 (11.12%)	227 (4.54%)	783 (15.66%)
Total Wins	2260 (45.2%)	2740 (54.8%)	5000
Total Points	2816	2967	5783

The analysis of the match between Fevga-2 and Fevga-3 (Table 2) gives some interesting information. The "human" strategy of Fevga-3 seems to win more games (54.8%). Nevertheless, the final result is almost equal, because Fevga-2 wins more double games (11.12% vs 4.54%). This is also confirmed after careful analysis of the results against Tavli3D: the two agents win the same number of games, but the Fevga-2 emerges superior because it wins more double games. We believe that the Fevga-2 strategy is better against weak opponents, because in the long run it wins more points than Fevga-3 due to more double games won. But when playing against a strong opponent, a little better strategy seems to be the more "human-like" strategy of Fevga-3, which maximizes total won games at the cost of doubles. Looking it from another perspective, we can say that the two versions have different playing styles: Fevga-2 plays more aggressively, trying to win more double games, while Fevga-3 plays more cautiously, paying more attention in securing the win than risking for doubles.

5.2 Experiments in Plakoto

Encoding the Raw Board Position. The input units of the neural network for the first version of Plakoto were the same 194 units of Fevga-1 plus 24 binary units for every player that indicated if the player had pinned a checker of his opponent at each point of the board. Thus, there were 242 input units in total. The agent with this coding

scheme was named Plakoto-1. As in Fevga, after only a few thousand games Plakoto-1 easily surpasses in strength the Tavli3D benchmark program. This improvement finally reaches a peak performance of about 1.15ppg (Fig.4. Left).

Using manual play, the level of Plakoto-1 was assessed as average by human standards. Strong aspects were the understanding of the value of pinning the opponent, especially in the home board. At the same time, it was also careful not to leave open checkers, thus not giving the opponent the chance to pin, because it understood that this will greatly increasethe chances of losing. Mistakes occurred often however, when it had to select a move that left at least one checker open: it did not take into account the possibility of the opponent pinning the open checker in the next move, thus rejecting moves resulting in positions with little or no chance for the opponent to pin and preferring moves resulting in open checkers very easily pinned.

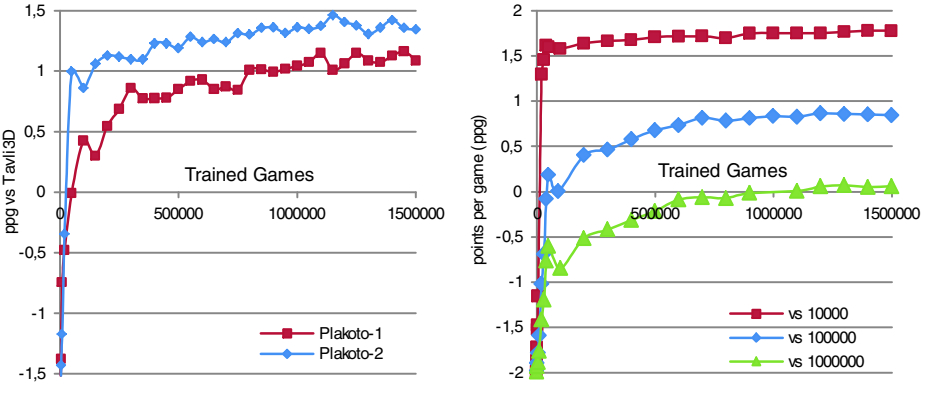

Fig. 4. Left. Training progress of Plakoto-1 and Plakoto-2 against Tavli3D. **Right.** Training progress of Plakoto-2 against stored weights at 10000, 100000, and 1000000 games trained.

Adding Expert Features. The following single feature was able to increase the performance considerably: the 24 binary inputs representing the pins of the opponent at each point were replaced by the probability of the opponent pinning a point if an open checker of the agent exists. This probability starts at 0/36 = 0 when no dice roll exist that can pin the open checker or no open checker exists, and maxes to 36/36 = 1 when all dice rolls pin the open checker or the checker is already pinned. This added feature required no additional input units as it utilized units already used by the neural network, only a little more computational overhead for computing the pinning probabilities. The resulting agent was named Plakoto-2. Compared to Plakoto-1, Plakoto-2 achieved better peak performance by about 0.3 ppg against Tavli3D (Fig.4.Left).A match of 5000 games between the two agents resulted in a comfortable win for Plakoto-2 (6687-1771, +0.98 ppg), further confirming the superiority of Plakoto-2. The level of Plakoto-2 was assessed as that of an experienced player.

Fig.3.Right and Fig.4.Right show the training progress of Fevga-3 and Plakoto-2 against previously stored weights. In both figures we see that the initial strategy improves rapidly for the first few thousand games and then improves more slowly to its peak performance.

6 Conclusion

This paper has shown that Reinforcement Learning combined with artificial neural networks is capable of producing high performance game playing programs in backgammon variants Fevga and Plakoto. In both games we used expert features to enhance performance. In the Fevga variant this has not led to much improvement compared to the raw features. We exploited a unique characteristic of Fevga, the existence of a good strategy according to the experts, to train a program with human-like playing strategy. Results showed that the completely different style of Fevga-2 is at the same level to the human-like strategy of Fevga-3. In the Plakoto variant, the addition of a single expert feature resulted in much better performance. In the future, we intend to further improve the playing strength of the agents by adding more features, and by introducing minimax search possibly using cutoff algorithms such as the ones described in [3].

References

1. BackGammon Variants, http://www.bkgm.com/variants
2. van Eck, N.J., van Wezel, M.: Application of reinforcement learning to the game of Othello. Computers and Operations Research 35(6), 1999–2017 (2008)
3. Hauk, T., Buro, M., Schaeffer, J.: *-minimax performance in backgammon. In: van den Herik, H.J., Björnsson, Y., Netanyahu, N.S. (eds.) CG 2004. LNCS, vol. 3846, pp. 35–50. Springer, Heidelberg (2006)
4. Schaeffer, J., Hlynka, M., Vili, J.: Temporal Difference Learning Applied to a High-Performance Game-Playing Program. In: Proceedings IJCAI, pp. 529–534 (2001)
5. Sutton, R.S.: Learning to predict by the methods of temporal differences. Machine Learning, 9–44 (1988)
6. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Indroduction. MIT Press, Cambridge (1998)
7. Szepesvári, C.: Algorithms for Reinforcement Learning (Electronic Draft Version) (June 2010), http://www.sztaki.hu/~szcsaba/papers/RLAlgsInMDPs-lecture.pdf
8. Tavli3D, http://sourceforge.net/projects/tavli3d
9. Tesauro, G.: Practical issues in temporal differnce learning. Machine Learning 4, 257–277 (1992)
10. Tesauro, G.: Programming backgammon using self-teching neural nets. Artificial Intelligence 134, 181–199 (2002)
11. Tesauro, G.: Temporal Difference Learning and TD-Gammon. Communications of the ACM 38(3), 58–68 (1995)
12. Veness, J., Silver, D., Uther, W., Blair, A.: Bootstrapping from Game Tree Search. Advances in Neural Information Processing Systems 22, 1937–1945 (2009)