The Significance of Temporal-Difference Learning in Self-Play Training TD-rummy versus EVO-rummy

Clifford Kotnik CLKOTNIK@ATT.NET
Jugal Kalita KALITA@PIKESPEAK.UCCS.EDU
University of Colorado at Colorado Springs, Colorado Springs, Colorado 80918

Abstract

Reinforcement learning has been used for training game playing agents. The value function for a complex game must be approximated with a continuous function because the number of states becomes too large to enumerate. Temporal-difference learning with self-play is one method successfully used to derive the value approximation function. Co-evolution of the value function is also claimed to yield good results. This paper reports on a direct comparison between an agent trained to play gin rummy using temporal difference learning, and the same agent trained with co-evolution. Co-evolution produced superior results.

1. Introduction

The success of TD-gammon is well known (Tesauro 1992 1995). Tesauro trained an artificial neural network (ANN) to approximate the value function for the game of backgammon without explicit expert advice programmed into the agent. With only the definition of legal moves and a reward when the game was won, temporal difference (TD) learning and self-play allowed the ANN to be trained well into the level of experienced human play. Further refinements allowed TD-gammon to reach expert level (Tesauro 1995).

TD-gammon was developed based on some of the early work on TD learning that has more recently been formalized and expanded. See, for example, Sutton and Barto (1998). The major challenge in deriving the value function is that there are many steps the agent must take before the game is won and a reward can be assigned. TD learning provides a method to assign credit from the reward to steps leading up to it. This is done in such a way that the value function can be adjusted in incremental steps as the game progresses. Combined with an ANN, this approach provides an error signal that is back-propagated at each step of the game to incrementally train the network. The algorithm differs from normal back-propagation in that the history of weight changes over the course of the game is used at each step. Sutton and Barto refer to this history as the *eligibility trace*.

There are those who question the significance of TD learning claimed based on the success of experiments such as TD-gammon. Pollack, Blair and Land (1996) argue that a simple co-evolutionary approach to deriving the weights for an ANN that approximates the value function works quite well. They argue TD learning is not the major reason for TD-gammon's success, and they suggest it is due to the self-play approach and specific features of the game of backgammon. Pollack, Blair and Land describe an experiment designed to mimic TD-gammon, but with the weights of the ANN derived by a simple evolutionary approach. However, the actual configuration for TD-gammon was not available to them, making direct comparison impossible. They tested against a publicly available version of Tesauro's backgammon player and reported encouraging results.

For this experiment, TD and evolutionary techniques are compared directly on the same agent with only the method of deriving the weights for the value approximation function differing. This allows the resulting players to play against each other. In addition, the cost of training can be directly compared.

2. Problem Definition

2.1 Game Definition

The problem is to train an agent to play the game of gin rummy. Gin rummy is a two-handed card game that can be summarized as follows (Gibson 1974):

- Deck: standard 52 card deck
- Rank: King=high, Ace=low
- Points: King, Queen, Jack=10; Ace=1; all others=face value
- Deal: 10 cards to each player; next card forms *discard pile*; remaining cards form the *draw pile*; discard pile is always face-up; draw pile is face-down; winner of each hand deals the next
- Goal: form meld from sets of 3 to 4 cards of same value or sequences of 3 or more cards of same suit and with the total of the face value

Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), Washington DC, 2003.

of remaining cards not so formed (called *deadwood*) less than or equal to 10; a single card cannot form part of a set and a sequence in the same hand

- Turn: during each turn a player can take the top card from the discard or draw pile, must discard one card face-up on the top of the discard pile and, if the goal state is reached, may lay down meld and deadwood (called *knocking*)
- Play: players alternate turns starting with the dealer's opponent until one player knocks
- Laying off: after one player knocks, the opponent may extend any of the knocking player's sets or sequences (called *laying off*) with any of his/her deadwood.
- Score: player who knocks scores the difference between the other player's deadwood points and his/her own. If the player who knocks has no deadwood, the other player is not allowed to lay off, and the player knocking receives a score of 25 plus the other player's deadwood points. If, after laying off, the opposing player's deadwood points are equal or less than the player knocking, the opponent scores 25 plus the difference in points instead of the player knocking.

A couple of simplifications to the game have been made for this experiment. We decided not to incorporate laying off in this experiment. Play usually continues until one player reaches 100 points. This portion of the game and other details of assigning bonus points beyond the description of a single hand are ignored for this analysis.

2.2 Reinforcement Learning Problem

The learning agent represents a gin rummy player, hereafter called simply "the player". The environment consists of the opposing player, the random sequence of cards on the draw pile, and the known sequence of cards in the discard pile. The game state is represented by the location of each card from the point of view of the agent. The state may be in-player's-hand (IPH), in-opponent's-hand (IOH), in-discard-pile (IDP) or unknown (UNK). A card is only considered to be IOH if it has been drawn from the discard pile. All other cards in the opponent's hand are considered to be UNK.

Gin rummy represents a moderately complex game that certainly cannot have all state-action combinations enumerated. With 52 cards in one of four possible states there are 4^{52} or approximately 2×10^{31} possible states. On the other hand it has a simple set of rules and small set of actions at each turn.

The actions that the agent can perform are to exchange any card in its hand for the top card of the discard pile, to exchange any card for the top of the draw pile or to take either of the preceding actions followed by knocking. The immediate reward is the score following a knock. It will be positive if the player scores, and zero if the opposing player scores.

The problem is episodic; each hand is an episode. The details of scoring multiple hands into a game total are ignored.

At the conclusion of each turn, a player has 10 cards. The value for this state is approximated with a function, as described below. During each turn, the player must decide whether to draw from the discard pile or the draw pile. Then the decision must be made as to which card to discard. Finally, the player must decide whether to knock.

The policy used is as follows: For the top card on the discard pile and for each card whose location is unknown (*i.e.*, possible cards on the top of the discard pile) the maximum of the value function is determined. This is accomplished by evaluating the hand resulting from exchanging each possible new card for each of the 10 cards currently in the player's hand. The value function is used to approximate each of these states. If the value function for drawing from the discard pile is greater than 50% of the possible cards on the draw pile, the player will pick from the discard pile, otherwise from the draw pile. The card to be discarded will be the one that leaves the maximum expected value for the remaining 10 cards. Note that this is one of the calculations already complete. If the total of the player's remaining deadwood is 10 or less, the player will knock.

The task is to learn the value function based only on the results of self-play. Except for the value function, the details of the above policy are implemented in discrete program logic. The value function has only the input of the game state and generates a single numeric evaluation of it. There is no knowledge of the rules of the game built into the value function. There is no notion of sequences, sets or deadwood.

3. Implementation

The implementation and experimentation phases were constrained to a fixed period of time. A limited number of machines were available to run the training. Given the CPU intensive nature of this sort of training, certain compromises were made to limit the training time. These will be identified in the following description.

An ANN is used to estimate the value function. The learning task is then to determine the weights for this network. The ANN is a feed-forward, multi-layer network with 52 inputs (one input for each card), 26 hidden units and a single output representing the value of that state. All layers of the network are completely

connected. Values for the four card states (IPH, IOH, IDP and UNK) are chosen with consideration for the network's use of the Euclidean distance between the states. IPH=2, IOH=-2, IDP=-1 and UNK=0. The activation function for the hidden and output units is the sigmoid. The game score is scaled to fall within [0,1] to correspond with the sigmoid values.

The approach used for inputs represents the accessibility of the card to the player. A positive value represents possession. Negative values represent increasing levels of inaccessibility. Zero is used for unknown. An alternate choice of ANN inputs for the card states is to have four binary or bipolar inputs for each card representing the states IPH, IOH, IDP or UNK. Only one of these inputs would be active at a time. While four inputs per card may provide a more accurate representation, it will also increase the network size by a factor of four and the run time will increase accordingly. We decided to use the faster representation for these tests.

The learning approach utilizes self-play. The opponent is implemented as a second copy of the player. The opponent has full knowledge of the cards in its hand and partial knowledge of the cards in the player's hand. Thus the player and opponent compete on an equal footing. This approach provides an endless set of training data, and it allows direct comparison of the TD and evolutionary players. Similar to the approach taken by Pollack et al. (1996), where the backgammon board was reversed, each game is played twice. First the cards are shuffled and dealt with one player going first. Then the same starting order is used and the other player goes first.

An epsilon-greedy approach is not implemented. It is assumed that the random order of the cards from the draw pile will generate enough exploration.

The Stuttgart Neural Network Simulator (SNNS) software package is used for ANN processing. The value function approximation is obtained by a forward pass through the SNNS network. A custom back-propagation algorithm built on top of the SNNS framework accomplishes training of the weights.

3.1 Temporal Difference Learning: TD-rummy

The approach used is taken from the TD-Gammon experiment as described in Tesauro (1992) and further explained in Sutton and Barto (1998). During each game, the player's value function is approximated by a forward pass of the game state through the player's network. Once the player makes its decision, the network is trained with the custom back-propagation algorithm developed for TD-rummy. Training continues incrementally after each turn the player takes.

The formula used for back-propagation training of the ANN weights, θ, is

$$\theta_{t+1} = \theta_t + \alpha(r_{t+1} + \gamma V_t(s_{t+1}) - V_t(s_t))e_t$$

where $\mathbf{e}_t$ is the vector of eligibility traces (Sutton and Barto 1998) that is built up over the course of each game based on the formula

$$e_t = \gamma\lambda e_{t-1} + \nabla_{\theta_t} V_t(s_t)$$

The immediate reward, r_{t+1}, is zero except when the player wins the game. In this case it is the score scaled to be in [0,1]. Like TD-Gammon, there is no discounting of rewards. $\gamma = 1$

Both the player and its opponent have their own network that is trained simultaneously with this algorithm. After an epoch of six games, the network corresponding to the player with the most wins is duplicated and trained for both players during the next epoch. The six games are actually three pairs of games with an identical starting state, but with the player and opponent reversed.

3.2 Evolutionary Learning: EVO-rummy

The second learning algorithm is the simple evolutionary approach described in Pollack and Blair (1996). The player and opponent start out using two random networks. They play an epoch, consisting of two pairs of games. If the opponent wins three of the games, the weights of the player's and opponent's networks are crossed by moving the player's network weight 5% in the direction of the opponent. If the opponent wins two or fewer games, the player's network is left unchanged. In either case, the opponent's network weights are mutated by adding Gaussian noise to them. The noise has a standard deviation of 0.1.

This approach implements a simple hill climbing. While this is called evolutionary, there is not a population of individuals that compete and go through selection for a new generation. There are only two individuals. When the opponent is measured to be superior, the evolving network is moved in that direction. Like Pollack and Blair (1996), I chose not to move too aggressively toward the opponent, moving just 5% of the way. The idea is to implement a simple approach. If a simple evolutionary approach can meet or exceed the temporal-difference technique, it provides more reason to question the significance of the latter.

Some experimentation with the training parameters is included in the results that follow. One network was trained with a crossover of 10%. Another network was switched to a higher threshold for crossover – five out of six games.

More details on both implementations will be available later this year (Kotnik 2003).

4. Results

Both learning algorithms were run on three to five different self-play configurations for a period of three weeks. Five two-processor Intel systems were used that contain GHz class CPUs. Both algorithms are CPU

intensive, with the TD algorithm using more CPU per game, as expected. In total more than 89,000 training games were played using the evolutionary approach, and 58,000 with the TD approach.

Name	Algorithm	Training Games	Decription
TD1	temp diff	9,484	alpha=0.1, lambda=0.3
TD2	temp diff	16,200	alpha=0.2, lambda=0.7
TD4	temp diff	16,243	alpha=0.2, lambda=0.2
TD5	temp diff	20,698	alpha=0.2, lambda=0.9
TD6	temp diff	1,800	alpha=0.2, lambda=0.9
EVO2	evolution	23,762	crossover=5%, mutation=0.1
EVO3	evolution	18,407	crossover=5%, mutation=0.1
EVO4	evolution	41,154	crossover=10%, mutation=0.1

Figure 1. Summary of the training parameter for the players whose performance is further analyzed.

The evolutionary algorithms were completed first, primarily due to the simpler software involved. Training commenced with the evolutionary algorithms while the temporal-difference software was coded. The number of turns required to reach a winning state with the initial random starting weights is very high. Games lasting 10,000 turns or more are not uncommon. Based on the initial tests with the evolutionary algorithms, a limit on the number of turns was introduced into both algorithms. Once reached, the game was considered a draw. For the majority of games, the limit was 5,000 turns.

Figure 1 summarizes the training parameters for the players whose performance is further analyzed. The learning rate, α, is in the range [0.1,0.3]. The temporal difference step weighting factor, λ, used is in the range [0.2,1.0]. Tesauro (1992) used 0.1 for α and 0.7 for λ. The larger number of games for evolutionary training is a result of the earlier completion of that software and hence a longer time to train. However, as will be described below, TD5 and EVO3 were the two "best of breed" players. Here the temporal-difference agent had more training games than the evolutionary agent.

The average number of turns per game turned out to be a useful approximation of the training progress. Figure 2 shows the progression of training for two players with each algorithm. This metric is only an early approximation as indicated by player EVO3, whose turns per game increased after 15,000 training games. On the surface, the value is rather disappointing since it never drops below 200. As anyone who plays gin rummy knows, the deck seldom has to be turned over to complete a game.

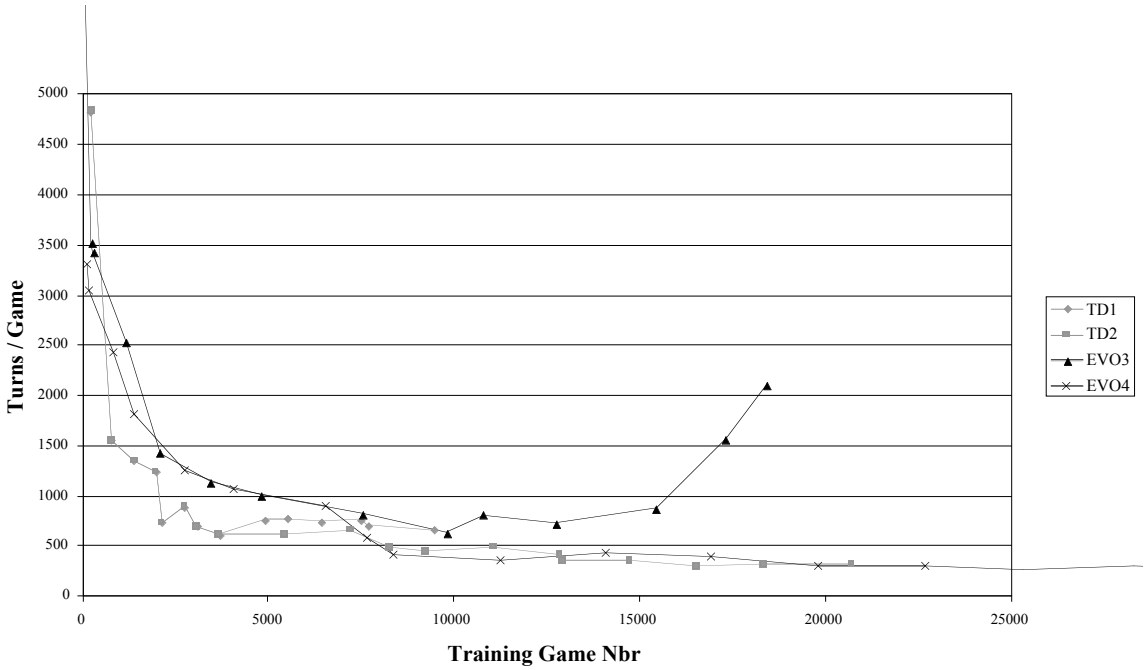

Figure 2 . Training turns per game.

The real measure of performance of a player is whether it wins against competitors. A tournament where each player played every other player for 10 games was conducted. The 10 games were five pairs of games with the deal reversed. Figure 3 contains the results of the three evolved players and four TD players. The RANDUMB player is a network of random weights such as that used to start all the training algorithms. Thus each player played 80 games. The 18 games not accounted for were draws.

Games Won — **Winner**

Loser	**EVO2**	**EVO3**	**EVO4**	**TD1**	**TD2**	**TD4**	**TD5**	**TD6**
EVO2		5	7	1	1		4	1
EVO3	5		2		1		2	
EVO4	3	8		1	2	4	6	4
TD1	9	10	9		7	7	6	4
TD2	9	9	8	2		2	4	3
TD4	10	10	6	3	8		8	3
TD5	6	8	4	4	6	2		2
TD6	9	10	6	2	7	6	5	
RANDUMB	10	10	10	10	10	10	10	10
Total Wins	**61**	**70**	**52**	**23**	**42**	**31**	**45**	**27**

Game Score — **Winner**

Loser	**EVO2**	**EVO3**	**EVO4**	**TD1**	**TD2**	**TD4**	**TD5**	**TD6**
EVO2		128	180	34	17		80	13
EVO3	65		32		12		59	
EVO4	84	136		13	38	36	67	31
TD1	392	424	394		257	248	243	154
TD2	405	300	365	15		85	158	83
TD4	350	422	218	70	292		336	159
TD5	221	363	201	199	242	48		38
TD6	395	342	162	35	217	215	187	
RANDUMB	474	445	442	399	420	421	432	419
Total Score	**2386**	**2560**	**1994**	**765**	**1495**	**1053**	**1562**	**897**

Figure 3. Tournament results of 10 games between players. Upper table shows the games won. Lower table shows the score.

As expected, the random weight lost every time. The other obvious fact is that the evolved players did much better on the whole than the TD players. The best overall player is EVO3 that won 87.5% of its games. This is the same player that showed the upswing in turns per game during training. The best TD trained player, TD5, had the largest value of $\lambda = 0.9$. TD6 used $\lambda = 1.0$, but was only trained for a short time.

One slight asymmetric aspect of the two training approaches is that of game score versus games won. The temporal-difference approach trained the network to estimate the game score. However, the evolutionary approach determined the best fit based on the games won. Therefore, figure 3 shows both metrics. Measuring performance base on score or games won produced the same ranking of players.

Winner

Loser	**EVO2**	**EVO3**	**EVO4**	**TD1**	**TD2**	**TD4**	**TD5**	**TD6**
EVO2		18	86	17	52		45	38
EVO3	189		89		18		33	
EVO4	122	22		30	34	50	37	45
TD1	34	18	30		1159	521	415	259
TD2	21	22	15	1275		579	72	147
TD4	32	20	22	586	1357		170	150
TD5	19	15	19	263	862	76		34
TD6	23	22	36	138	175	911	895	
RANDUMB	151	39	42	72	57	101	52	60

Figure 4. Average turns per game in tournament.

Figure 4 contains the average turns per game for the tournament. These results indicate that the similarly trained players play the same and tend to prolong the game.

The unexpected results that EVO3 outperforms the other evolved players that have longer training cycles may indicate over-training. To test this, a tournament was conducted not only with the final version of the players, but also with intermediate results. Figure 5 shows the results for three players from each algorithm after various regimes of training. These regimes correspond to roughly 5-10,000 games.

EVO4 encountered a serious problem after regime 5. TD4 has stagnated. The other players do not show definite signs of over-training. However, more data are needed to reach any firm conclusion.

A version of the tournament program presents the state of the game on an animated display showing each player's actions and the cards after each turn. Observing the play of a number of games, one technique the evolved players

developed that the TD players did not was to rapidly dispose of high cards in order to be able to knock quickly. This causes EVO-rummy players to interfere with each other in obtaining sets and sequences of smaller cards.

From observing games between EVO3 and TD5, it is clear that EVO3 is superior. Both players make obvious mistakes, but EVO3 makes far fewer. While neither player has reached a human level of play, you have only to watch the nearly endless progression of a game between players using random networks to see how far the training has come.

To further investigate the difference in strategy learned by these players, the cards in each player's hand at the end of each game in the final tournament were analyzed. The number of sets and sequences were summarized as were the number of cards of each suit and each face value. To quantify the affinity of the players for cards of certain face values or suits, a chi-squared test of the hypotheses that the players' final hands contained a even distribution of suits and face values was calculated. The results are shown in figure 6.

The EVO players' hands contained approximately twice as many sets as sequences. RANDUMB's hands contained the reverse, or twice as many sequences as sets. In addition to learning to go after sets and sequences, the EVO players also appear to favor cards of lower face value, which makes sense. The chi-squared tests are consistent with this, indicating a high probability the evolutionary players do not have a suit preference and a low probability they do not have a face value preference.

The TD players' hands contained almost all sequences. The TD players developed a strong affinity for a single suit. This seems to indicate the TD players just learn to go after one suit and the sequences just happen. The chi-squared tests support this.

Name	Chi-squared Suit	Chi-squared Face
EVO2	72%	30%
EVO3	50%	21%
EVO4	80%	29%
TD1	3%	100%
TD2	1%	100%
TD4	2%	100%
TD5	1%	100%
TD6	1%	100%
RANDUMB	31%	97%

Figure 6. Chi-squared test of the hypotheses that the players choose suits and face values evenly.

5. Conclusions

Self-play coupled with an artificial neural network to evaluate the value function for a game is an effective training method. This experiment has demonstrated that when self-play is coupled with a simple, evolutionary technique the result can outperform the more guided temporal difference learning technique. Both techniques, when combined with self-play, can train agents on a set of rules without explicit human training.

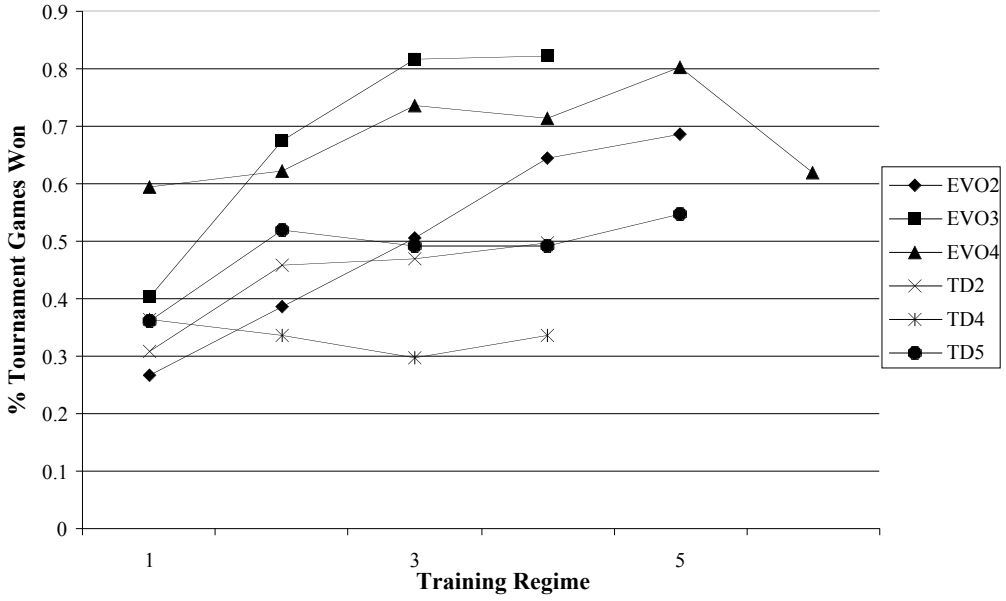

Figure 5 . Overtraining test.

In these experiments, the agents trained with the evolutionary approach developed a much more balanced strategy, going after sets and sequences and rightly favoring cards with lower face value. The temporal-difference training generated agents that simply collected a single suit. It appears that some mechanism is required to introduce more exploration in the temporal-difference training.

The general approach is not without difficulties. The whole process is extremely compute intensive. In addition, there are a number of parameters the experimenter must set that need to be compared at various levels. This only makes the long training times more problematic. Self-play can lead the agents to be trained on a specific type of play that may not generalize, as with the single suit approach for TD.

5.1 Further Research

The assumption that the TD training approach for this task does not require explicit exploration may not be valid. Further testing with an epsilon-greedy or similar approach added to the training will determine if additional exploration can improve performance.

Certainly an extended period of experimentation will be worthwhile. The basic impact of the temporal-difference parameters needs to be studied in a more systematic way. Tesauro (1992) even suggests that the learning rate be gradually decreased during the training process in something akin to simulated annealing. It has been suggested that gamma values slightly less than one might help.

The impact of the basic parameters for the evolutionary approach can be studied. However, there are a number of more sophisticated algorithms that can be tried as well.

Both approaches may benefit from several refinements. Changing the ANN topology with four inputs for each card may produce better results. In addition, more hidden units in the ANN may help. The basic training and measurement approach should be consistently set for either best total score or most games won. In the policy for determining whether to select from the draw or discard pile, it would be better to use the mean rather than median expected value.

Continued experimentation can certainly benefit from faster training times. The maximum turns before a draw is called might be lowered. Optimization of code to play the game and perform the back-propagation faster will help to speed the ability to experiment. With the number of players trained, there is only a hint of what impact the various parameters have. A more optimized training program can allow these to be explored. More training is also needed to determine if over-training is an issue.

It will be interesting to add the feature allowing a player to lay off cards when its opponent knocks. This will make the game more realistic and challenge the player to pay more attention to its opponent's cards.

References

Gibson, Walter. (1974). *Hoyle's Modern Encyclopedia of Card Games.* New York: Doubleday.

Kotnik, Clifford. (2003). Training Techniques for Sequential Decision Problems. *Forthcoming Master of Science Thesis*. University of Colorado at Colorado Springs.

Pollack J. B., Blair A. D. & and Land M. (1996). Coevolution of a Backgammon Player. *Proceedings of the Fifth International Conference on Artificial Life.*

SNNS Stuttgart Neural Network Simulator, University of Stuttgart and University of Tübingen, http://www-ra.informatik.uni-tuebingen.de/SNNS/

Sutton, Richard and Andrew Barto. (1998). *Reinforcement Learning, An Introduction.* Cambridge: MIT Press.

Tesauro, Gerald. (1995). Temporal Difference Learning and TD-Gammon. *Communications of the ACM*. Vol 38:3, 58-68.

Tesauro, Gerald. (1992). Practical Issues in Temporal Difference Learning. *Machine Learning*. Vol 8, 257-277