Board Representations for Neural Go Players Learning by Temporal Difference

Helmut A. Mayer
Department of Computer Sciences
Scientific Computing Unit
University of Salzburg, AUSTRIA
helmut@cosy.sbg.ac.at

***Abstract*— The majority of work on artificial neural networks (ANNs) playing the game of Go focus on network architectures and training regimes to improve the quality of the neural player. A less investigated problem is the board representation conveying the information on the current state of the game to the network. Common approaches suggest a straight–forward encoding by assigning each point on the board to a single (or more) input neurons. However, these basic representations do not capture elementary structural relationships between stones (and points) being essential to the game. We compare three different board representations for self–learning ANNs on a 5×5 board employing temporal difference learning (TDL) with two types of move selection (during training). The strength of the trained networks is evaluated in games against three computer players of different quality. A tournament of the best neural players, addition of α–β search, and a commented game of a neural player against the best computer player further explore the potential of the neural players and its respective board representations.**

Keywords: Game of Go, Artificial Neural Networks, Temporal Difference Learning, Board Representation

I. Introduction

With the advent of computers, board games have attracted many researchers, e.g., [1], as the computational intelligence of game playing programs can be directly related to the intelligence of its human opponent. Out of all board games, chess has received the most attention with efforts beating the human world champion finally being successful in 1997. (*Deep Blue*, a chess–playing IBM supercomputer, defeated Garry Kasparov, the reigning world champion in chess [1]).

The board game Go has received increasing attention in recent years, as unlike chess programs the best Go programs are still at a mediocre amateur level, i.e., a good amateur Go player easily beats the machine. The rule set of Go is very small, but the seemingly simple concepts build into deep and complex structures on the board. For an excellent and compact introduction to the game we refer to [2], and to [3] for computational aspects. Despite the simplicity of Go's rules, the game's strategies and tactics are difficult to put into analytical or algorithmical form. There are mainly three reasons why Go is hard for traditional computer game–playing techniques.

First, the number of possible moves (the branching factor) in the majority of game situations is much larger than in games such as chess or backgammon with about 20 legal moves for each board position. On a standard 19×19 Go board a player has the choice among 200–300 potential moves. Hence, in a common game tree representation, where each node is associated with a board situation and each branch with a move, the number of nodes grow exponentially with a base of 200. A Go computer program playing with a very moderate tree depth of four had to evaluate 10,000 times the number of moves a chess program has to ponder.

Second, Go is a game of mutual dependent objectives. While in chess the goal is very explicit (capture of the opponent's king), in Go the aim of securing territory (where each board intersection counts as a point) can be achieved by capturing opponent's stones (death) as well as by securing own stones (life). As a consequence, evaluation functions precisely assessing a board situation can hardly be defined, as human expert players often rely on rather intuitive concepts, e.g., *good* and *bad shape* (of stones). Hence, ANNs having been successfully applied in the field of pattern recognition are promising candidates to improve the quality of Go programs.

Third, though Go has been played for thousands of years in China and Japan, the first professional Go players started to earn prize money 45 years ago. Professional chess has a tradition of 130 years resulting in much more literature on opening, mid–, and end game theory based on millions of recorded games played by expert players. As a matter of fact, today's extremely strong chess programs rely on human expertise to defeat human expertise.

A radically different approach is the construction of computer players by exquisite learning from playing against opponents (computers and/or humans), or even against itself. Eventually, the programs improve their playing strength without any explicit incorporation of a priori knowledge, which gives these systems the potential to "invent" game strategies no human player has ever discovered.

The "star" among artificial board game players is Tesauro's (1995) neural backgammon player *TD–Gammon*. Based on *Temporal Difference* (TD) learning, a reinforcement learning technique, a network has been trained in self–play by only receiving feedback on the outcome of games. After millions of training games (in its latest version) TD–Gammon is estimated to play at a level extremely close to the world's best human players [4].

[1]http://www.research.ibm.com/deepblue/

The impressive performance of TD–Gammon inspired many researchers to employ TD learning with other board games including Go. Schraudolph et al. (2000) suggest a sophisticated network architecture and local reinforcement to train a network against a randomized version of *Wally* by Bill Newman on the 9×9 board [5]. The authors claim that their network beat the commercial program *Many Faces of Go* by David Fotland after 3,000 training games, however the skill level of the program was set to 2–3 out of 20 (best), and the game statistics do not show the number of wins, but the number of stones lost to the opponent.

Ekker (2003) presents TDL variants using different training algorithms to teach a network from play against *Wally* on a 5×5 board [6]. He found that TD–(μ) (a TD variant considering imperfect play of the opponent) [7] utilizing residual–λ learning achieved the best results. The author reports that networks having learned from Wally win 80 % against it, and close to 50% against GNU Go (version not given) at the lowest level (see GNU Go comments in Section III).

Evolutionary approaches are another way to generate neural Go players, e.g., [8], [9], [10], without human intervention. Here, networks are evolved against dedicated computer players, or in a coevolutionary manner by competition of evolving individuals. In a recent paper Runarsson and Lucas (2005) compare TD learning and coevolutionary approaches for Go on the 5×5 board [11]. The authors conclude that both techniques achieve a similar level of play, when using a linear weighted evaluation function. In games against a randomized version of GNU Go (v3.4), where with a probability of 0.5 a random move replaced GNU Go's choice, self–learnt and coevolved players won approx. 80% of the games.

II. TD Learning of Neural Go Players

Learning from self–play offers some appealing advantages to conventional ANN training. Even, if training yields an ANN player having extracted all the concepts hidden in the training data, it is very likely that it will never surpass the strength of the players, whose games constituted the training data. E.g., in [12] ANNs having been trained with chess games by master players, played reasonably against strong players, but failed to beat weak players.

Self–learning ANNs do not require any knowledge of the game, but only of the games' rules and feedback about the outcome of the game. Hence, in theory the neural player could have playing abilities beyond any human player, as it does not rely on human expertise at all. Nice as this may sound, there are practical limitations to self–learning, most prominently, the computational cost associated with self–learning and the large number of games necessary to sample the (in case of Go) extremely huge search space.

Hence, we restricted self–learning of neural Go players to the simple 5×5 board, which is mostly used for educational purposes and demonstration of basic concepts of the game. In terms of computational cost we believe that self–learning of Go players for a 9×9 board is the current limit (unless one spends months and years of CPU time).

The networks in this work are trained with the TD(0)–algorithm [13] given by

$$\vec{w}_{t+2} = \vec{w}_t + \eta[\gamma V_{t+2} - V_t]\nabla_{\vec{w}_t} V_t, \qquad (1)$$

where $\vec{w}$ are the weights of the network, V is the value of the selected action (move), r is the reward, η is the learn rate, and γ is the discount factor ($\gamma = 1.0$ in all of the following experiments). By multiplying the gradient of the value with the TD error (the term in square brackets), the value for the move selected at time step t is increased or decreased depending on the value of the best move at $t+2$. Thus, moves leading to higher values in subsequent time steps are reinforced, i.e., are trained to trigger a higher value themselves. As can be seen in Equation 1 the network mostly learns from its own predictions, but at the end of each game a reward r (1 for a win, 0 for a loss) substituting $V(t+2)$ gives the important feedback from the real world.

A. Temporal Difference and Reward Scheme

We would like to point out two important details of our implementation of TD(0) learning. The temporal difference is calculated between two moves (hence $t+2$ in Equation 1). Usually, the temporal difference of values before and after a single move of a player [4] is utilized, which gave poor results in our experiments. We believe that considering two subsequent moves can improve game learning in general, as the value $V(t+2)$ also incorporates the response of the opponent, and serves as an immediate feedback of the quality of a move. However, it should be stressed that the latter feedback, again, is "only" a prediction of the network, as it plays both colors, and thus may be wrong.

Another adjustment turned out to be even more important for reasonable play acquired by self–learning. In the manner described above reward is given to the player making the last move (always a pass move in Go, as the game is ended by two subsequent passes). If black was the last to move and won the game, then the net receives a reward of 1. The "message" given to the net is that it played well with the black *and* the white stones, when in fact bad white moves may have caused black's win. Consequently, black wins more and more games easily by reinforcement of white's bad play, which overall leads to a weakly performing neural player. Thus, in our implementation the network always receives two rewards. The first one as described above, and the second one is given to the opponent. Of course, both rewards are given to the same network with the same board (end) position (causing different inputs for different colors), but if black wins, the network also receives a reward of 0 for its white role, and vice versa.

B. Move Selection

We employed two variants of action (move) selection during learning, namely, the ϵ–greedy and a *Softmax* method [13]. With the ϵ–greedy approach the move with the highest value is selected, but with a small probability ϵ a random move is chosen instead so as to explore the search space. If a temporal difference value is affected by a random move, we omit the learning step, as a random move is not in accordance to the value estimations of the neural player. With a random move all moves (even the worst) have equal probability of being selected, a potential problem being alleviated by softmax methods. These select random moves by assigning higher probabilities to moves with higher values, e.g., *Gibbs Sampling* used in [5].

We devised a softmax method based on the *Creativity* factor c, which we introduced originally to add some variability to the play of a trained network. Typically, given a specific board situation a trained network will always play the same move (the one with the largest value). To avoid this rather mechanic behavior and to play different but reasonable (maybe even better) moves all moves with values in the range $[v_c, v_{max}]$ are considered with equal probability, where v_{max} is the largest value, and

$$v_c = v_{max}(1 - c) \qquad 0 \leq c \leq 1. \tag{2}$$

We utilize this technique for the softmax action selection variant, which we term *creative* move selection ($c = \epsilon = 0.05$ in all of the following experiments). Note that the number of creative moves may vary according to the stage of training. In early stages most move values are similar and close to the maximum, while in later stages a single move may be very important and its value makes it the only move being considered. Also moves, which are close to the best, will have a good chance to be explored more often, and may quickly be trained to maximal value, if they prove to be beneficial.

C. Board Representations

In this work we put our focus on the investigation of different board representations (Figure 1), as the simple representation often used for neural Go players does not convey information on the neighborhood of intersections on the Go board, even though, neighborhood is one of the most important concepts in Go. We term[2] this simple representation used in related work (e.g., [5], [11]), where each intersection is mapped to an input neuron with values depending on the stone (not) occupying the intersection, *Koten*[3].

With all three following representations we rather speak of own and opponent instead of black and white stones, as the same network may play both colors (certainly against itself). An own stone is encoded by a value of 1, an opponent stone by -1, and an empty intersection yields an input value of 0. Hence, in the specific example given in Figure 1 the koten value for the marked intersection is 1, if white is about to move, and -1, if it is black's turn (the marked intersection is occupied by a white stone). For the 5×5 board this results in 25 input neurons.

The *Roban* representation is inspired by the ANN input encoding of a checkers board in [14], where overlapping subsquares of different sizes covering the board were used. Here, we employ all 3×3 squares with different center positions. In Figure 1 the positions of a specific subsquare are shown. The mean value over all nine positions is the ANN input value ($\frac{-1}{9}$ for white, $\frac{1}{9}$ for black in Figure 1). Additionally, the mean value (differential) of the complete 5×5 board is fed into the network. This gives in total ten network inputs (nine values for different 3×3 squares).

The *Katatsugi* representation tries to capture the essential concept of neighborhood. An intersection is encoded by a weighted sum of the values of the intersection (center) and its four neighbors building a solid connection, i.e., katatsugi. We experimented with different weights for the center and neighbor positions, but performance differences were small. In all of the following experiments the weights for all five positions are identical with a value of 0.2. Note that with this representation the values of edge and corner points lie in a smaller interval than those of other intersections, which gives the network some ability to differentiate between types of intersections. The katatsugi value for the example in Figure 1 is 0 (for the weight given) regardless of the color the network is playing. As with koten, the network has 25 input values when katatsugi is used to encode the Go board.

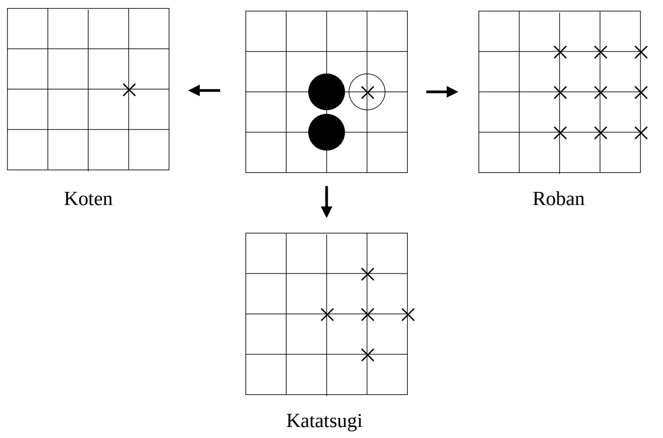

Fig. 1. Three board representations for the marked point in the top center board.

The single output neuron of the TD network gives the estimated value of the board position encoded by a specific representation at the input.

D. Performance Measure

In order to monitor the development of the self–learning Go players quantitatively we used the strength $s = \frac{w}{g}$ being the win rate (w is the number of wins) of a player challenging one or more Go players in a number of games g. In the following experiments (Section IV) the strength has been measured in games against three computer players (Section III) of different quality ranging from a pure random player to a heuristic player including search for common Go patterns on the board.

[2]In an attempt to honor the eastern origin of the game we use Japanese terms.

[3]`http://senseis.xmp.net/`

III. Computer Go Players

For the evaluation of the neural Go players we utilized three heuristic computer players of different playing abilities, which are briefly described in the following.

The *Random* player's only "knowledge" of the game is the ability to discern between legal and illegal moves, i.e., out of all legal moves (including the pass move) one is chosen randomly with uniform probability distribution. This player's main purpose is to detect very basic Go skills in a computer player, as a human novice with some hours of Go practice should easily beat the Random player. Also, it serves as a test for a neural player that possibly is able to win against a modest computer player, but does not have a general concept of Go, i.e., it may lose against Random.

The *Naive* player may be compared to a human knowing the rules of Go, and having played some games is familiar with basic concepts. It is able to save and capture stones, and knows when stones are definitely lost. Weak stones, i.e., stones in danger of being captured, are saved by connecting them to a larger group, so that a weak stone becomes a member of a living group (or at least of one with more liberties).

GOjen is a Go program written in Java largely based on Fuming Wang's program *JaGo*, and is the best computer player we have used. It knows standard Go playing techniques (saving and capturing stones), and searches the board for 32 well–known Go patterns and its symmetrical transformations. A few program errors have been fixed, and time performance has been increased considerably by the author.

In order to rate a Go player's strength there are ranking systems for amateur and professional players. The amateur ranking system starts with the student (*kyu*) ranks from 35 kyu up to 1 kyu (best). When an amateur becomes a master (*dan*) player, she gets the rank of 1 dan (best is 7 dan). Professional ranks being above all amateur ranks are on a scale from 1 to 9 dan.

We used the free Go program *GNU Go* [4] (version 3.2) with an estimated rank of 10 kyu to determine the strength of GOjen, and arrived at a rank of about 28 kyu. Thus, GOjen plays at the level of a beginning amateur player after some weeks of game practice.

Go on a 5×5 board has been solved [15]. Black wins with a score of 25 points (no komi), when playing the optimal opening move C3 (board center). Black also wins starting play with C2, C4, B3, and D3 (by a score of 3, no komi).

GNU Go optimally opens a game (C3) with the black stones on a 5×5 board, and with the white stones passes immediately after black C3, C2, C4, B3, and D3. As a self–trained ANN only has to learn the optimal opening move (which it mostly does) to win against GNU Go, this program has not been utilized in evaluating the strength of the neural players, but will definitely serve as a valuable opponent on larger boards.

[4]`http://www.gnu.org/software/gnugo/`

IV. Experiments

This section presents self–learning experiments of neural Go players employing feed–forward networks with a single output neuron representing the estimated value of the board position fed into the input layer.

A. Experimental Setup

For each of the three investigated board representations we run self–learning experiments utilizing both move selection methods, namely, ϵ–greedy ($\epsilon = 0.05$) and creative ($c = 0.05$) (Section II-B). The koten and katatsugi nets consist of 25 input neurons and 25 neurons in a single hidden layer (a total of 650 links). For a fair comparison the roban nets have 10 input neurons with two hidden layers each containing 20 neurons (a total of 620 links). The activation function of all hidden neurons is the sigmoid function. All other neurons have linear activation. The training algorithm is standard error back–propagation with a learn rate $\eta = 0.01$.

Each of the 20 training runs consists of 1 million games starting with a random network (weights in $[-1.0, 1.0]$). The strength of the networks is sampled every 50,000 games by playing 1,000 games each (half black/white, no komi) against GOjen, Naive, and Random (Section III). In these evaluation games moves are selected strictly according to maximal value.

B. Self–learning Results

In Figure 2 the strength development of neural players is depicted for the three board representations utilizing ϵ–greedy move selection.

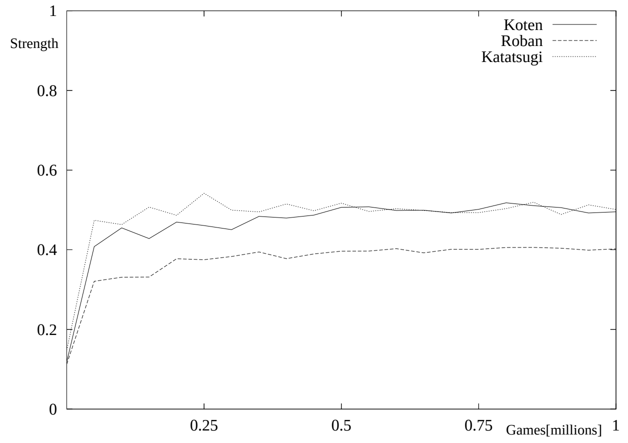

Fig. 2. Strength development during self–learning with ϵ–greedy move selection for different board representations (averaged on 20 runs).

The katatsugi nets exhibit the best performance until they are caught by the koten nets at around 500,000 games settling in at a strength of 0.5. The roban nets are clearly inferior to both other representations and reach a strength of 0.4 after 1 million games. All three representations reach their best performance level at around half a million games and stagnate from there on.

In Figure 3 the strength development of the players learning with the three different board representations and creative move selection is shown.

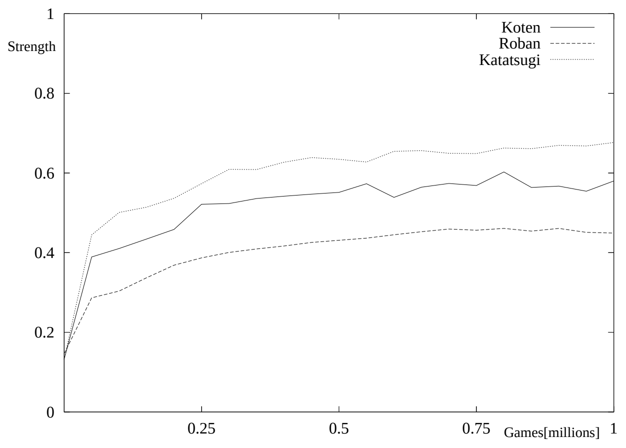

Fig. 3. Strength development during self–learning with creative move selection for different board representations (averaged on 20 runs).

Clearly, all three representations are separated in strength with katatsugi winning the race with a strength of 0.68 after 1 million games. The koten nets reach their best level after 800,000 games at a strength of 0.6 as do roban nets at 0.46. All representations benefit of the creative move selection shown by considerably greater strength (compared to ϵ–greedy), and especially katatsugi shows some potential for further improvement by additional training games.

In Figure 4 a game between GOjen (black) and the best creative katatsugi net of all 20 runs (white, strength 0.7693) is shown.

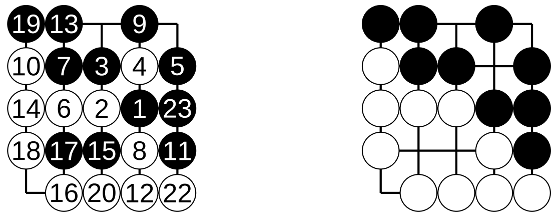

Fig. 4. The best creative katatsugi net (playing the white stones) wins against GOjen.

GOjen opens the game with the near–optimal move 1 (Section III) answered by the net's optimal move 2. At this point the network estimates its chances to win with 0.68. Net's move 4 is questionable, but it makes GOjen "think" that it has to attack 4 immediately with 5, which allows net to play the important move 6. Net's 8 forces the program to capture 4 with 9. The moves 10 to 14 are solid standard moves by both players. By playing 15 the program correctly takes its last chance to win the game "hoping" for an error of the neural player. E.g., 18 played in the left, lower corner or at 20 would immediately loose the game for white. Though, this is quite obvious for a human player, we often noticed that trained networks fail to "see" important and game–saving moves. However, Katatsugi(g) convincingly ends the game (being slightly optimistic to have won, estimate 0.61) with a score of 1.0 (each player has three points of territory, but white has captured two stones, black only one).

C. Tournament of Neural Players

Finally, we compare the best networks generated in the various experiments by performing a round robin tournament among them. Each competitor plays 1,000 games (500 each color) against each other. Note that though, the networks play deterministically (creativity set to 0), it is possible that two networks play different games, if in a specific board situation two or more moves have the same maximum value (then, among these a random move is drawn).

For each of the three board representations we selected the network of greatest strength generated in all greedy and creative learning runs, respectively. The scores in Table I are the win percentages of all games a net has played.

TABLE I
TOURNAMENT OF BEST GREEDY (G) AND CREATIVE (C) NETWORKS.

Rank	Score	Best Net
1	0.7040	Koten (c)
2	0.5892	Koten (g)
3	0.4866	Katatsugi (c)
4	0.4796	Katatsugi (g)
5	0.3758	Roban (c)
6	0.3282	Roban (g)

Interestingly, here the koten nets beat the katatsugi nets. This is mainly based on the fact that koten(c) beats koten(g) in every single game (playing 0.5 against katatsugi(c)), and katatsugi(c) loses every game to katatsugi(g). Hence, certain weaknesses of networks are fully exploited by others. However, for each board representation the creative network outperforms the greedy network, and the roban nets are clearly inferior to the others.

D. Deep Search

Finally, we present the strength and the performance against the three computer players of the best creative katatsugi net, when utilizing α–β search with depths of one (simple search), two, and four moves.

TABLE II
PERFORMANCE OF BEST CREATIVE KATATSUGI NET WITH VARIOUS SEARCH DEPTHS.

	Katatsugi(c)		
Depth	1	2	4
GOjen	0.6500	0.5830	0.6560
Naive	0.7370	0.8920	0.9110
Random	0.9860	1.0	1.0
Strength	0.7527	0.7753	0.8320

As can be expected the strength increases with search depth, however, variations can be observed in games against GOjen. This shows that not all board situations are evaluated correctly by the neural player, as it occasionally comes up with weaker moves after deeper search. The best creative koten net being a bit weaker at depth one (strength 0.7033) gains more by increasing the search depth (0.7890 at two, 0.8293 at four) arriving at a comparable level.

V. Summary and Conclusions

We have presented self–learning experiments of neural Go players based on temporal difference learning (TDL) on a 5×5 board investigating three different board representations and two variants of move selection methods, namely, the well–known ϵ–greedy method and our suggested softmax method termed creative move selection. The strength of the neural players has been evaluated in games against three different computer players ranging from a pure random player to a naive player having some elementary Go knowledge and a more sophisticated but still weak player with an estimated rank of 28 kyu.

It could be shown that the creative move selection proliferates considerably better players when compared to the ϵ–greedy method. This may be attributed to the fact that the creative method samples the search space in promising regions more densely, as it does only explore moves whose value is within a small range (depending on the creativity parameter) of the best move. Thus, it is able to more quickly identify moves, which are superior to the move currently estimated to be the best. Also, the simple katatsugi board representation, which captures essential characteristics of the board structure and key concepts of the game showed its potential in combination with the creative move selection. On average the katatsugi nets outperformed the networks based on the other two basic representations (roban and koten). Subjectively, when observing play the katatsugi nets are more aware of basic and very important capture and save moves requiring knowledge of structural context.

All self–taught networks exhibited a consistent and robust style of play demonstrated by win rates being inverse proportional to the quality of the computer players. Especially, the fact that the trained networks beat the random player at rates of approx. 98% show that the networks indeed learn general game concepts and do not learn only specific sequences of moves. This may also be credited to small improvements in our TDL implementation (Section II-A). The best networks achieved a win rate of 65% against the best computer player GOjen, and always learned to play the optimal opening move in the board center. This win rate compares nicely to our work on evolution of neural players [10], where networks evolved against GOjen beat the program in 68% of the games. However, it should be stressed that the evolved networks have been specifically adapted to the program, whereas the self–trained nets in this work did never face the program during training.

As better computer programs, e.g., GNU Go, can only be effectively used as an opponent on larger boards, our next step will be the investigation of the presented methods on 7×7 and 9×9 boards. In order to decrease the computational cost (self–learning in 20 million games on 5×5 took approx. 70 hours on a 2.13GHz processor under Java/Linux) we are currently working on methods to transfer the knowledge incorporated in the 5×5 nets on larger boards. Also, we are exploring techniques to combine evolutionary and self–learning approaches.

References

[1] C. E. Shannon, “Programming a computer for playing chess,” *Philosophical Magazine*, vol. 41, pp. 256–275, March 1950.

[2] C. Chikun, *Go: A Complete Introduction to the Game*. Kiseido Publishing Company, 1997.

[3] M. Müller, “Computer Go,” *Artificial Intelligence*, vol. 134, no. 1–2, pp. 145–179, 2002.

[4] G. Tesauro, “Temporal Difference Learning and TD–Gammon,” *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, March 1995.

[5] N. N. Schraudolph, P. Dayan, and T. J. Sejnowski, “Learning to Evaluate Go Positions via Temporal Difference Learning,” IDSIA, Tech. Rep. 05–00, February 2000.

[6] R.-J. Ekker, “Reinforcement Learning and Games,” Master’s thesis, Rijksuniversiteit Groningen, 2003.

[7] D. Beal, “Learn from your opponent – but what if he/she/it knows less than you?” in *Step by Step*, J. Retschitzki, Ed. Editions Universitaires Fribourg Suisse, 2002, pp. 123–132.

[8] N. Richards, D. Moriarty, P. McQuesten, and R. Miikkulainen, “Evolving Neural Networks to Play Go,” in *Proceedings of the 7th International Conference on Genetic Algorithms*, 1997.

[9] A. Lubberts and R. Miikkulainen, “Co–Evolving a Go–Playing Neural Network,” in *2001 Genetic and Evolutionary Computation Conference Workshop Program,*. San Francisco: Morgan Kaufmann, July 2001, pp. 14–19.

[10] H. A. Mayer and P. Maier, “Coevolution of Neural Go Players in a Cultural Environment,” in *Proceedings of the Congress on Evolutionary Computation 2005*. IEEE Press, September 2005.

[11] T. P. Runarsson and S. M. Lucas, “Coevolution Versus Self–Play Temporal Difference Learning for Acquiring Position Evaluation in Small–Board Go,” *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 628–640, December 2005.

[12] S. Thrun, “Learning To Play the Game of Chess,” in *Advances in Neural Information Processing Systems 7*, G. Tesauro, D. Touretzky, and T. Leen, Eds. Cambridge, MA: MIT Press, 1995, pp. 249–252.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.

[14] K. Chellapilla and D. B. Fogel, “Evolving an Expert Checkers Playing Program without Using Human Expertise,” *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 4, pp. 422–428, 2001.

[15] E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk, “Solving Go on Small Boards,” *International Computer Games Association Journal*, vol. 26, no. 2, pp. 92–107, 2003.