Competitive Reinforcement Learning for Combinatorial Problems

Conference Paper · February 2001

DOI: 10.1109/IJCNN.2001.938727 · Source: IEEE Xplore

Myriam Abramson
United States Naval Research Laboratory
32 PUBLICATIONS **223** CITATIONS

Competitive Reinforcement Learning for Combinatorial Problems

Myriam Abramson
George Mason University
Fairfax, VA 22030-4444
mabramso@gmu.edu

Dr. Harry Wechsler
George Mason University
Fairfax, VA 22030-4444
wechsler@cs.gmu.edu

Abstract

This paper shows that the competitive learning rule found in Learning Vector Quantization (LVQ) serves as a promising function approximator to enable reinforcement learning methods to cope with a large decision search space, defined in terms of different classes of input patterns, like those found in the game of Go. In particular, this paper describes S[arsa]LVQ, a novel reinforcement learning algorithm and shows its feasibility for Go. As the distributed LVQ representation corresponds to a (quantized) codebook of compressed and generalized pattern templates, the state space requirements for online reinforcement methods are significantly reduced, thus decreasing the complexity of the decision space and consequently improving the play performance. As a result of competitive learning, SLVQ can win against heuristic players and starts to level off against stronger opponents such as Wally. SLVQ outperforms S[arsa]Linear when playing against both a heuristic player and Wally. Furthermore, while playing Go, SLVQ learns to stay alive while SLinear fails to do so.

1 Introduction

Game playing has been dominated by game-tree search[Russell and Norvig, 1995] using look-ahead heuristic evaluation. The high-degree of interactions between moves, combined with the high branching factor, however, makes tree-search methods unsuitable for games like Go. Tactical and strategic considerations are difficult to integrate in a heuristic evaluation. Tactical considerations are based on local move patterns. Strategic considerations are based on whole board evaluation. The relative urgency of those moves is best determined on a case-to-case basis by the outcome of the game. Reinforcement learning can address the problem of action (*move*) selection based on delayed rewards and has been successfully used in Backgammon[Tesauro, 1995]. For longer games such as Go, one has to cope with increased spatio-temporal decision complexity in addition to space complexity as it is the case for Backgammon. The claim of this paper is that for a large decision search space, defined in terms of different classes of input patterns, like that found in the game of Go, the competitive learning rule found in LVQ is a promising function approximator for reinforcement learning methods. Towards that end, this paper describes SLVQ and shows its feasibility for Go.

2 The Game of Go

Go is a perfect information, deterministic, 2-player game usually played on a 19x19 board. It can also be played on smaller boards of arbitrary sizes. The board is empty at the beginning of the game and the players alternate placing stones on the board. Stones are placed on the intersection of lines. Once placed on the board, a stone never moves unless captured. The 4 adjacent points of a stone on the board are called “liberties” of the stone. Connecting stones in a group increase the number of liberties. A stone is captured when it has no liberties left. The aim of the game is to enclose territory or vacant points. The only way to achieve this goal is to make your stones “alive”. Here, alive is taken in the weak sense meaning that your stones can’t be captured. It has been found that the capability to make a 2-eye shape (Figure 1) is sufficient and necessary for life. There are 3 distinct phases of the game corresponding to different patterns of play: opening, middle game, and endgame. [1]

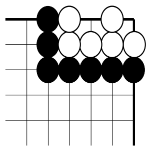


Figure 1: The white stones are alive in a 2-eye shape.

The Complexity of Go

Since Go is a deterministic, perfect information game, perfect play should be possible if not for the large search space. How hard is Go and what is its relevance to machine learning? Figure 2 shows the different peaks of potential solutions, only one of which, the key move, will lead to a winning game. The smoothness of the landscape indicates the absence of correlation between the evaluation function, in this case the sum of influence values, and the distance to

[1] A good starting point for more information is www.usgo.org.

the goal, a characteristic of complex problems. Go is also hard for machine learning because there are no identifiable features. Rather each feature is a pattern that, like fractals, grows into larger and more detailed patterns.

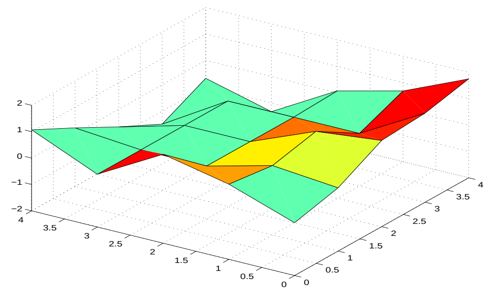
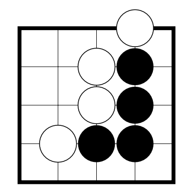


Figure 2: Fitness landscape of candidate moves as the sum of influence values for the board in Figure 3

Representation and Evaluation of the Board

The board is represented as a square matrix of stone objects. A list of legal moves is generated and passed to the players. A legal move does not commit suicide, does not repeat a previous position[2], and does not put a player in atari (similar to check in chess)[3], but single throw-ins are allowed and kos are recognized. The evaluation of the board is done using the Chinese rules which allow a player to fill its territory without penalty[4] and a program to play without recognizing life or death patterns. No komi points were added for white and no prisoners were included in the final score. After each move, the influence value of the move (+1.0 for black and -1.0 for white) is propagated to neighboring stones. This representation was first used by Zobrist[Zobrist, 1970]. In our implementation, each neighbor (including diagonals), if it's an empty point, computes its influence value by summing the influence value of its neighbors (8-connected neighbors) with equal weight until no changes occur in a ripple-like fashion. The influence value representation conveys the relationships between the stones (Figure 3). At the end of the game, the difference in territory won (over the maximum territory) is propagated as the reward. This somewhat simplified version of the game still retains its main strategic aspects.



-0.72	-0.73	-0.49	-1.00	0.13
-0.7	-0.74	-1.00	1.00	0.4
-0.73	-0.6	-1.00	1.00	0.87
-0.62	-1.00	1.00	1.00	0.93
-0.59	-0.17	0.34	0.84	0.92

Figure 3: 5x5 pattern and its influence representation

[2]Zobrist's hashing.

[3]Although this last constraint is not part of the rules of Go and does not allow the common "throw-in" tactic, it makes it easier to define a reasonable "pass" move.

[4]We used Robert Jasiek's rules at http://home.snafu.de/jasiek/simple.html.

3 Learning

This section describes briefly the reinforcement learning and self-organization learning principles required for the SLVQ Go algorithm (see Section 4).

Reinforcement Learning

The interactive nature of reinforcement learning is particularly appealing for game learning since the early days of Samuel's checker player[Samuel, 1990]. What is learned through interaction with the environment is an optimal policy mapping states to actions maximizing the total reward obtained. Compared to other learning paradigms, reinforcement learning has some nice properties for game learning: no expert knowledge is needed and it is incremental, that is continuous learning against a variety of opponents is possible. From a reinforcement learning point of view, Go is a finite-horizon sequential-task problem with discrete states. Because of the large state space, it is helpful to view the game with continuous states using the influence value propagation of the stones.

Policies for Reinforcement Learning

There are two basic ways of using experience in reinforcement learning: off-policy and on-policy. They differ only by the update rule used to arrive at an optimal policy. The off-policy, embodied in the Q-learning algorithm[Watkins, 1989], uses the estimate of the optimal policy for update of the existing policy and consequently separates exploration from control. The on-policy, embodied in the Sarsa algorithm[Sutton and Barto, 1998], uses the current estimate of an existing non-optimal policy for refinement towards a *better* existing policy. The only guarantee to arrive at an optimal policy with Sarsa is possible only if the existing policy progressively inches itself towards an optimal policy. In both policies, convergence has been proved in the discrete, tabular case if each action is selected infinitely often.

An on-policy approach for game learning has more opportunities for active learning in the exploration of moves as well as better on-line performance in *teaching* games. In addition, credit assignment to previous moves, or eligibility trace, is not limited as it is the case for the off-policy approach where greedy control moves have to coincide with the exploratory policy. Better informed exploratory policy will be the subject of future research.

Self-Organization and LVQ

Self-organization involves the ability to learn and organize (cluster) sensory information without the benefit of a teacher. Learning is driven by measures of fitness, possible evolved over time. If the task to be learned is that of clustering, one example of such a fitness measure is that of similarity. The process of self-organization consists of iteratively modifying synaptic weights in response to sensory patterns until an optimal configuration, according to some closeness measure, eventually develops. One particular class of self-organizing systems that are of interest to us are the Self-Organizing Feature Maps (SOFM) [Kohonen, 1997], which are driven by competitive learning. In the competitive learning scheme, the output neurons of the network compete among themselves to be activated or fired, with the result that only one output neuron or one neuron per group is on at any one time. The locations of the winning neurons tend to become ordered with respect to each other in such a way that a meaningful lattice like coordinate system eventually emerges and faithfully represents the sensory input.

There are many situations where the clusters derived as a result of self-organization have to be appropriately labeled as it would be the case for information retrieval. Towards that end, one expands SOFM using a supervised learning scheme as it is the case with Learning Vector Quantization (LVQ). In the case of LVQ, the labeled clusters collection correspond to a (quantized) codebook of compressed pattern templates.

The LVQ algorithm[Kohonen, 1997] is a supervised clustering method in which each output unit represents a particular class or category. The weight vector for an output unit is often referred to as a reference or codebook vector for the class that the unit represents. It is also assumed that a set of training patterns with known class labels is provided, along with an initial distribution ("seed") of reference vectors. After training, the neural net classifies an input vector by assigning it to the same class as the (labeled) output unit that has its weight vector closest to the input vector. After learning, the probability density function of the input is approximated by the modified set of discrete decoders or codebook vectors. The distributed representation of LVQ into codebook vectors as generalization of the input patterns significantly reduces the state space requirements and has a close correspondence to a tabular representation of state-action pairs.

4 The Sarsa Learning Vector Quantization (SLVQ) Player

SLVQ integrates Sarsa with LVQ. This integration loosely ties the estimation of the utility function Q to the pattern recognition task of the situation. In an on-policy control algorithm, the utility of the *next* move taken a' will be used to update the state-action pair $Q(s,a)$ (for the off-policy control algorithm, the utility of the *best* move according to the current estimate of the optimal policy is the one used). The update of the weight vectors is a function of the change in the utility of the move. Let $\Delta Q(s_t,a)$ be the change in utility for state s at time t and associated action a, the prototype vector m that matched s_t most closely then moves closer to or away from the input vector x accordingly:

$$s_t \sim m_t$$
$$\Delta Q(m_t,a) = \alpha_m \lambda_{m_t} \left[\gamma Q(m_{t+1},a') - Q(m_t,a)\right]$$
$$m(t+1) = m(t) + \Delta Q(m_t,a)\left[s_t - m_t\right]$$

and where a' is the next action taken under the policy followed in an on-policy approach. The choice of an on-policy approach gives us better on-line performance at the expense maybe of less flexibility in exploration. The learning rate α is local to m and decays proportionally as a function of the number of times m won the competition. Algorithm 1 shows the backward implementation view of SLVQ. In addition, learning is also possible in this context from the opponent's action to provide a balance of positive and negative examples to the learning task.

Algorithm 1 SLVQ

```
Initialize $\gamma$ and $\lambda$.
Initialize C $\leftarrow \{\overrightarrow{m},a,Q,\alpha,e\}_i$ , the initial set of codebook, action, utility, learning rate and eligibility tuples.

REPEAT for each episode; *or each game*
    Initialize $e$=0 for each tuple
    clear history

    REPEAT for each step of episode; *each move of the game*
        $\overrightarrow{s_t} \leftarrow$ current state
        normalize $\overrightarrow{s_t}$
        $C_t \leftarrow \arg Softmax_C$ similarity($\overrightarrow{s_t},\overrightarrow{m}$)
        decay $\alpha_t$
        $C_{t-1} \leftarrow$ previous match
        $\delta = \gamma Q_t - Q_{t-1}$
        $e_{t-1} = 1.0$
        FOR each state $t-i$ in history *;backup*
        $Q_{t-i} \leftarrow Q_{t-i} + \alpha_{t-i}\delta e_{t-i}$
        $\overrightarrow{m_{t-i}} \leftarrow \alpha_{t-i}\delta e_{t-i}(\overrightarrow{s_{t-i}} - \overrightarrow{m_{t-i}})$
        normalize $\overrightarrow{m_{t-i}}$
        $e_{t-i} \leftarrow \gamma\lambda e_{t-i}$
        END

    UNTIL end of episode

    $C_{lastmatch} \leftarrow$ last state in history

    $\delta = reward - Q_{lastmatch}$

    $e_{lastmatch} = 1.0$

    FOR each state $t-i$ in history
        $Q_{t-i} \leftarrow Q_{t-i} + \alpha_{t-i}\delta e_{t-i}$
        $\overrightarrow{m_{t-i}} \leftarrow \alpha_{t-i}\delta e_{t-i}(\overrightarrow{s_{t-i}} - \overrightarrow{m_{t-i}})$
        normalize $\overrightarrow{m_{t-i}}$

    END

UNTIL no more episodes
```

The initialization of the weight vectors is done from real games played against Gnugo[5]. Each board configuration

[5]http://www.gnu.org/software/gnugo/gnugo.html

makes up a weight vector after the propagation of the influence values and normalization. The softmax exploration policy was used with SLVQ where action selection is implemented by adding a small random number decaying with time to the action evaluation and picking the action with the largest sum[Sutton and Barto, 1998].

Matching

Matching of a board configuration against a weight vector is done using the fuzzy constrast model [Santini and Jain, 1999]. This distance is not a metric distance but takes into account the presence or absence of certain stones in a pattern. This characteristic makes it extremely well suited for the recognition of the vital stones in a game. Let a pattern P be represented by the set of influence values p_{ij} (after normalization) at each of its points. The similarity S between patterns P and R can be computed as a function of their commonality and their reflexive differences:

$$Com(P,R) = \sum_i \sum_j min\{p_{ij}, r_{ij}\}$$
$$Diff(P,R) = \sum_i \sum_j max\{p_{ij} - r_{ij}, 0\}$$
$$S(P,R) = Com(P,R) - \alpha Diff(P,R) - \beta Diff(R,P)$$

α and β represents the relative saliency of the prototype and the variant (input pattern). The saliency of a board pattern P represented as a set of quantitative features by the influence values of the stones can be determined as the density $\sum_i \sum_j w_{ij} p_{ij}$ where w_{ij} is proportional to the inverse distance relationship of the move associated with the pattern. Rotation and mirror symmetry of the board is performed to compute a best match. Figure 4 shows the codebook vectors becoming a closer approximation of the input space as training progress.

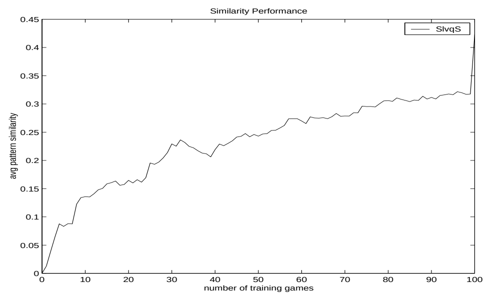


Figure 4: Similarity of the codebook vectors

Additional codebook vectors were added from the sample input when no good match was found, that is when the similarity score was negative.

Final action evaluation was done by computing the pareto optimality of moves along the pattern similarity dimension and the utility (Q) dimension. This prevents the selection of codebook vectors with good similarity but poor outcome.

5 Experimental Results

This section presents experimental data to show the feasibility of SLVQ Go and its comparative merits compared to other potential Go players.

Go Players

Wally Wally[Millen, 1981] is a low-level player but its moves are surprisingly good despite no knowledge of the “whole” board situation. It uses simple heuristics: 1) capture; 2) put in atari; 3) search the board for a pattern matching a table of “good” moves ordered by urgency and breaking ties randomly; and 4) if nothing else was found to do, play a random move. The program[Newman, 1988] was modified to play white as well as black and to play different board sizes.

“Heuristic” This player deterministically chooses the move generating the highest sum of influence values. This evaluation is consonant with the input representation of the board. This heuristic has much more strategic value than tactical value.

SLinear This player implements the Sarsa algorithm with the equivalent of a linear function approximator. Each point on the board is a feature with 3 possible values: black, white or empty. Weights, initialized to 1.0, are associated with each feature and the policy is to pick the move with the greatest weight. This player learns *good* moves by playing random games. The exploration and the learning rate decay proportionally as the square root of the number of games played. This player is a TD(λ) version of Monte Carlo Go[Brugmann, 1993]

Experimental Setup

The parameters were initialized as follows:

5x5
$\lambda = 0.5$
$\gamma = 0.9$
$\alpha = 0.05$ decays as a function of $\frac{1}{\sqrt{h}}$ where h is the number of times a codebook vector has won the competition.
exploration = 0.05

7x7
$\lambda = 0.9$
$\gamma = 0.9$
$\alpha = 0.09$ decaying as above.
exploration = 0.09

Results after Training

The offline results below show the average territory difference lost over 5 games for SLVQS (SLVQ with softmax as the exploration strategy) and SLinear playing as Black against the “Heuristic” player or Wally. Training was done against the same opponent as the games played offline. Please note winning corresponds on the average to negative territory lost. The player for whom the average territory lost is positive but small in value, however, occasionally wins some games. Smoothing of the graphs (10%) ensures a easier readability of the performance trend in this complex game where one move difference can drastically alter the outcome of the game.

5x5 * A 5x5 game has approximately 13M possible configurations under rotational invariance. The legal move constraints bring this figure down somewhat but the state space still remains huge. The codebook vectors were initialized with sample boards from 27 games for a total of 331 vectors. Figures 5 and 6 show the comparative performance of SLVQS and SLinear. SLVQS routinely defeats the “Heuristic” player and defeats Wally most of the time too. One can see that SLVQS outperforms SLinear.

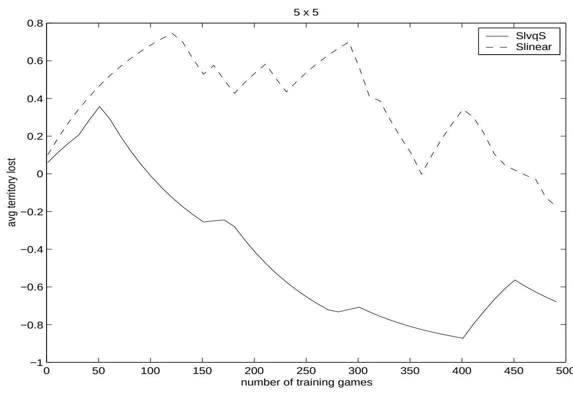


Figure 5: 5x5 games against the “heuristic” player

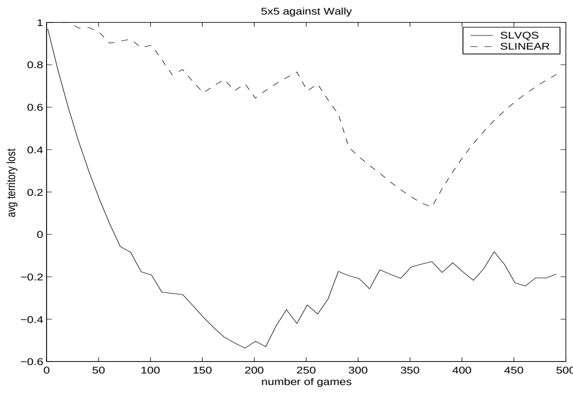


Figure 6: 5x5 games against Wally

Figure 7 shows the results of SLVQS and SLinear against Wally in the problem of Figure 3. SLinear does not learn to live as the average territory lost quickly converges to 1. SLVQS levels against Wally towards the end of the training.

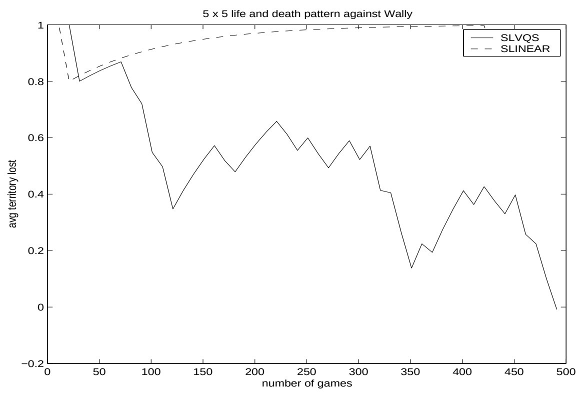


Figure 7: 5x5 pattern in Figure 3 against Wally

7x7 * Increasing the dimensions of the board from 5x5 to 7x7 makes the game clearly more difficult. The codebook vectors were initialized from samples of 42 games or 659 codebook vectors. Figure 8 shows results against Wally. The performance of SLVQS improves as training progresses. SLVQS will win a few games against Wally and again SLVQS outperforms SLinear.

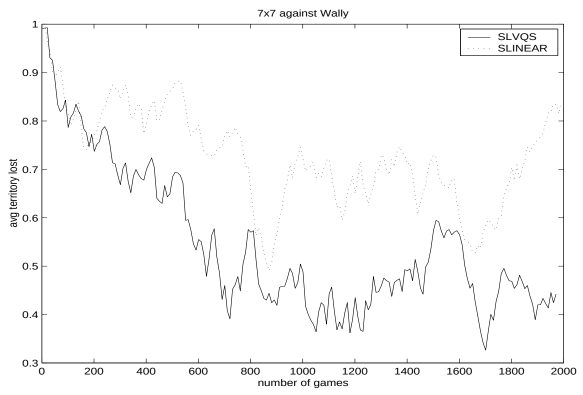


Figure 8: 7x7 games against Wally

Figure 9 shows SLVQS winning against the “Heuristic” player. Clearly, it is easier to learn a strategy against a deterministic player and harder to learn to play against a pure tactical player since SLVQ has only a generalized representation of the board and here again, SLVQS outperforms SLinear.

Figure 11 shows results on the life-and-death pattern of Figure 10. Similar to the 5x5 pattern, SLinear does not learn the goodness of certain moves in special circumstances. SLVQS again does much better than SLinear. As training goes on it attempts to level off against Wally and occasionally wins some games. There seems to be a plateau occurring in the performance of SLVQS which might indicate some amount of overfitting.

SLVQ plays a good opening (Figure 12). It knows the value of the center on a small board and of the corners. It knows

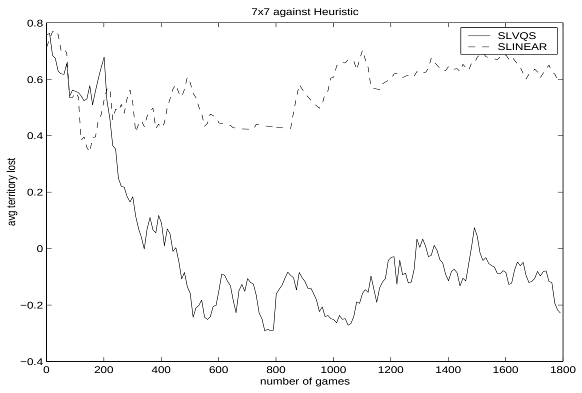


Figure 9: 7x7 games against “Heuristic”

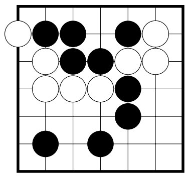


Figure 10: 7x7 life-and-death pattern

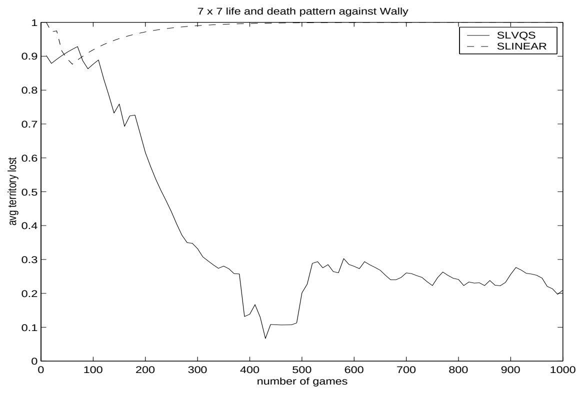


Figure 11: 7x7 life-and-death pattern of Figure 10 against Wally

to cut and put a stone in atari but also has some amount of senseless *throw-ins* which are necessary in the endgame to finish the game under the program’s rules. Overall, SLVQS plays too aggressively and tries to capture first rather than defend even when the capture was not urgent. Training specifically for those non-obvious moves on Go problems might be necessary.

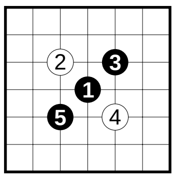


Figure 12: SLVQS (black) opening against Wally

6 Conclusions and Future Work

This research shows the feasibility of learning a cognitive skill with reinforcement learning in a knowledge-free approach. Beyond Go, other application domains include, for example, navigational tasks and C^3I. The distributed representation of LVQ reduces the state space significantly when the input space can be decomposed into different classes. The integration of LVQ and Sarsa enriches the reinforcement learning paradigm to include competitive learning. Future research will include enhancement of the exploration strategy, further decomposition of the input space to localized regions of the board, and learning action coordination with a gating network.

References

[Brugmann, 1993] Brügmann, B., “Monte Carlo Go”, Monte Carlo Go `ftp://www.joy.ne.jp/welcome/igs/Go/computer/mcgo.tex.Z`, 1993.

[Kohonen, 1997] Kohonen T., *Self-Organizing Maps*, 2nd Edition, Springer, 1997.

[Millen, 1981] Millen J.K., Programming the Game of Go, *Byte Magazine*, April 1981.

[Newman, 1988] Newman,W.H.,available at http://ftp.nuri.net/Go/computer/wally.sh.Z, 1988

[Russell and Norvig, 1995] Russel S. J., Norvig P, *Artificial Intelligence: a Modern Approach*, Prentice Hall, 1995.

[Samuel, 1990] Samuel, A. L., Some Studies in Machine Learning using the Game of Checkers, reprinted in *Readings in Machine Learning*, Morgan Kaufman, 1990.

[Santini and Jain, 1999] Santini, S., Jain R., Similarity Measures, *Transactions on Pattern Analysis and Machine Intelligence*, Vol 21, No 9, September 1999.

[Sutton and Barto, 1998] Sutton, R., Barto A., *An Introduction to Reinforcement Learning*, MIT Press, 1998.

[Tesauro, 1995] Tesauro, G, Temporal Difference Learning and TD-Gammon, *Communication of the ACM*, Vol. 38, No. 3, 1995.

[Watkins, 1989] C. Watkins, *Learning from delayed rewards*. Ph.D. Thesis, King’s College, 1989.

[Zobrist, 1970] Zobrist, A. L., *Feature Extractions and Representation for Pattern Recognition and the Game of Go*, PhD dissertation, Graduate School of the University of Wisconsin, 1970.