EVALUATION IN GO BY A NEURAL NETWORK USING SOFT SEGMENTATION

M. Enzenberger
University of Alberta, Edmonton, Alberta, Canada
emarkus@cs.ualberta.ca, http://www.cs.ualberta.ca/~emarkus/

Abstract In this article a neural network architecture is presented that is able to build a soft segmentation of a two-dimensional input. This network architecture is applied to position evaluation in the game of Go. It is trained using self-play and temporal difference learning combined with a rich two-dimensional reinforcement signal. Two experiments are performed, one using the raw board position as input, the other one doing some simple preprocessing of the board. The second network is able to achieve playing strength comparable to a 13-kyu Go program.

Keywords: Go, neural networks, segmentation, connectivity, NeuroGo

1. Evaluating Go Positions

Writing a program that plays the game of Go is a notoriously hard problem. Despite many efforts the best programs still play at a weak to medium amateur level of about 8 kyu (Schaeffer, 2001). This is not only due to the large branching factor but also to the fact that the evaluation of Go positions is difficult. State-of-the-art programs rely on a knowledge intensive approach. They use large databases of patterns, rule-based systems, and hand-tuned heuristics (Bouzy and Cazenave, 2001).

1.1 Simplification by Segmentation

The evaluation of a Go position can be simplified by segmenting the position into parts. This works well in positions with independent subgames where playing one subgame does not affect the value of other subgames. A typical example are Go endgame positions to which combinatorial game theory has been applied successfully (Müller and Gasser, 1996). Positions without a clear segmentation are more difficult: in particular, middle-game positions with many possible continuations each leading to different follow-up segmentations, and positions with multiple nearby tactical fights. Many Go programs use some influence-based segmentation of positions (Chen, 2002).

H. J. Van Den Herik et al. (eds.), *Advances in Computer Games*

Cognitive studies on human Go players have shown that humans perceive Go positions not as a set of hierarchical structured patterns with clear boundaries but rather as a set of overlapping clusters (Reitman, 1976).

1.2 Neural Networks

Neural networks have been used for evaluating full-board Go positions. Schraudolph, Dayan, and Sejnowski (1994) used temporal-difference learning (Sutton, 1988) to train a neural network to evaluate Go positions. They showed that it is important to use a rich reinforcement signal and a sparsely connected network architecture that reflects the local character and translational invariance of the pattern-recognition task. This was accomplished by using 5×5 receptive fields with weight sharing.

However, essential features of a Go position depend on whether two points on the board are connected by one colour or will become connected later. Using fixed-size receptive fields makes the recognition of long distance connections impossible. The most basic cases are *blocks*. Blocks are sets of adjacent stones of the same colour; they can take an arbitrary shape on the board. Moreover, they can only be captured as a unit.

The Go program NEUROGO (version 2) used receptive fields that dynamically adapt their size to fit around blocks (Enzenberger, 1996). This was achieved by transforming the Go position into a graph with all stones of a block merged into a single node. While NEUROGO's performance was improved greatly compared to a network using fixed-size receptive fields, it was still impossible for the network to represent higher-level objects like *groups*. Groups are a set of loosely connected blocks that might become connected later. This was the motivation for the development of a new network architecture with better abilities for segmenting the board.

2. Architecture using Soft Segmentation

This section presents a neural-network architecture that is able to process a Go position by building a soft segmentation of the position. This architecture is now used in version 3 of NEUROGO.

The neural network uses a feedforward backpropagation architecture. The neurons have a sigmoid activation function, with activation values between 0 and 1 and a bias weight. The soft segmentation of a position is represented as two connectivity maps, one for each colour. Each connectivity map assigns a connectivity strength between 0 and 1 to each pair of points on the board. See Figure 1 for an overview of the network architecture.

The next section describes the reinforcement signal that is used for learning, followed by a description of the layers in the network and the connections between them.

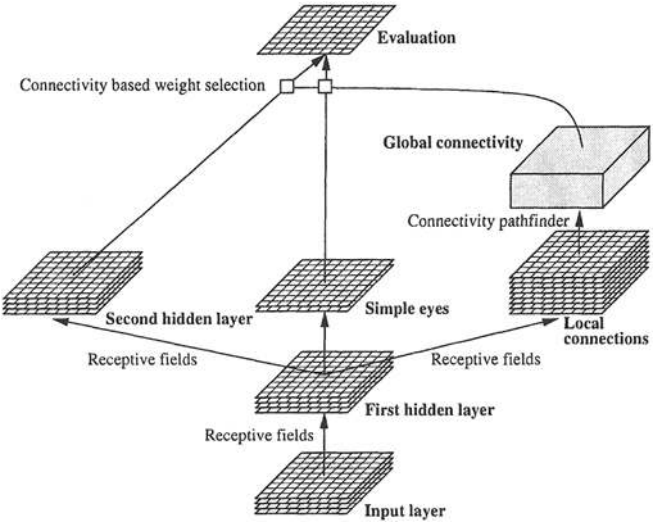


Figure 1. Network architecture.

2.1 Reinforcement Signal

The final position of a Go game contains much richer information than merely the global score. The network uses single-point eyes, connections, and live points which are defined as follows.

- A *single-point eye* is an empty point with all adjacent points occupied by stones of the same block or by stones of two blocks of the same colour that share another single-point eye.
- A pair of points is *connected* by one colour if there is a path between them containing only stones of that colour or single-point eyes.
- A point is said to be *alive* if it is connected to two single-point eyes.

Chinese scoring rules are used during the network training. No pass move is allowed until all points on the board are alive[1]. Also, it is not allowed to play in one's own single-point eyes.

Single-point eyes and connections can occur in earlier positions of the game, but may not exist in the end position, because those blocks could have been captured. Live points stay alive from the first position in which they occur until the end position.

The network uses single-point eyes, connections and live points as a reinforcement signal. Connections are used only locally within a 3×3 window centred around each point.

[1]This makes scoring and detection of the end of the game easy, but will lead to wrong play in case of seki situations or more complicated single-point eyes (involving more than two blocks). However these cases rarely occur in actual games.

2.2 Neuron Layers

Each layer of neurons in the architecture contains one or more neurons for each point on the board. There are 7 layers as follows.

Input layer: This layer contains one or more neurons per point depending on the number of (boolean) input features that are used. The activation of the neurons is set to 0 or 1 according to whether a certain input feature is present in the Go position at this point. A Go position is always transformed such that Black is to move. This makes an additional input for indicating what colour is to move unnecessary.

First hidden layer: This layer contains one or more neurons per point. The number of neurons per point is a parameter of the network architecture. The layer is connected with receptive fields to the input layer.

Second hidden layer: Like the first hidden layer, this layer contains one or more neurons per point. The number of neurons per point is another parameter of the network architecture. The layer is connected with receptive fields to the first hidden layer.

Simple eyes layer: This layer contains 2 neurons per point, one for each colour. The activation is a prediction of whether that colour is able to create a single-point eye at this point. The layer is connected with receptive fields to the first hidden layer. It receives a reinforcement signal when a simple eye is created on the board.

Local connections layer: This layer contains 18 neurons per point, 9 for each colour. The activation is a prediction of whether that colour is able to create a connection from this point to each of the 9 points in a 3×3 window around this point (including self-connection). Neurons corresponding to off-board points are unused. The layer is connected with receptive fields to the first hidden layer. It receives a reinforcement signal when a connection is created on the board.

Global connectivity layer: This layer contains $2 \cdot n^2$ neurons per point for board size n. The activation is a prediction whether each colour is able to create a connection from this point to any point on the board. The activation is computed by the connectivity pathfinder (see 2.5) from the local connections layer.

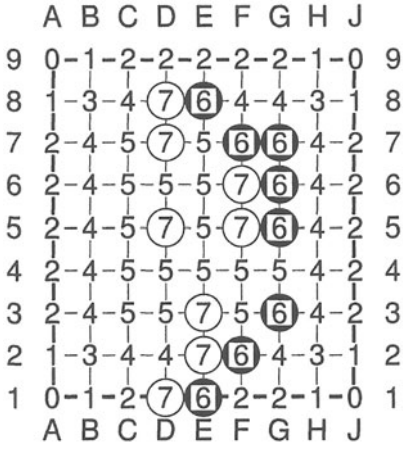
Evaluation layer: This layer contains 1 neuron per point. The activation is a prediction whether this point will be alive for Black (activation 1) or White (activation 0). The layer is connected to the second hidden layer and the simple eyes layer by connectivity-based weight selection (see 2.6). It receives a reinforcement signal for live points when they are created on the board.

2.3 Point Types

Each neuron corresponds to a point. Different point types are defined. The actual weights are chosen from weight sets depending on the point type.

There are two reasons for using weight sets. They increase the number of free parameters without significantly affecting the time for processing a position, since only one weight of a set is selected. They also compensate for effects of the edge of the board while still making it possible to learn local patterns that are mostly invariant with respect to translation.

The function type(p) assigns a type to each point p. See Figure 2 for the point types that were used.



type(p)	
0	Empty corner point
1	Empty edge point next to corner
2	Other empty edge point
3	Empty point diagonal from corner
4	Other empty point on second line
5	Empty point on line 3 or higher
6	Black stone
7	White stone

Figure 2. Point types: definition and example.

2.4 Receptive Fields

The function window(p) assigns to each point p the set of points within a 3×3 square window centred at this point. If a layer is connected with receptive fields to a previous layer then each neuron corresponding to a point p is connected to all neurons in the previous layer corresponding to the points $p' \in \text{window}(p)$. The spatial relationship of two points p and p' is described by a field index given by the function field(p, p') (see Figure 3).

Consider a layer L with n neurons per point connected to a previous layer L' with m neurons per point by receptive fields. Then a neuron corresponding to a point p and index $i \in \{1..n\}$ is connected to all neurons in the previous layer corresponding to points $p' \in \text{window}(p)$ and index $j \in \{1..m\}$ using the weights

$$w^{LL'}_{i,j,\text{type}(p),\text{type}(p'),\text{field}(p,p')}$$

The neuron has a bias weight $b^{L}_{i,\text{type}(p)}$.

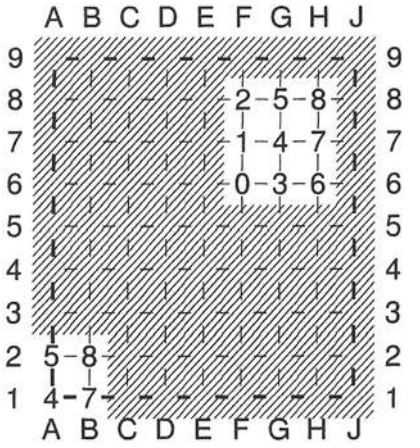


Figure 3. Receptive fields. Field indices for 2 receptive fields centred at A1 and G7.

2.5 Connectivity Pathfinder

The connectivity pathfinder creates a global connectivity map from the local connections layer. It assigns a connection value between 0 and 1 to each pair of points for each colour.

Local connections are assumed to be independent. Connection values of points outside the local connection window are computed as the product of the local connection values. Only the path resulting in the highest connection value is considered. The current implementation of the pathfinder runs Dijkstra's shortest-path algorithm with each point as a starting point.

2.6 Connectivity-based Weight Selection

The simple eyes layer and second hidden layer are connected to the evaluation layer using connectivity-based weight selection.

Every neuron in the evaluation layer is connected to all neurons in the previous layer with weights depending on the connection value between the corresponding points predicted by the global connectivity layer. For that purpose connection values are transformed from the continuous values between 0 and 1 into 8 equally sized intervals[2]. For each colour c and pair of points p and p' the function $\text{connection}(c, p, p') \in \{1...8\}$ returns the index of the interval.

Consider a neuron corresponding to a point p in the evaluation layer E. The neuron has a bias weight $b^E_{\text{type}(p)}$. Let L be one of the previous layers to which the evaluation layer is connected by connectivity-based weight selection (the simple eyes layer or second hidden layer), with n neurons per point. Then the

[2]For efficiency, points with a connection value smaller than 0.1 were ignored.

neuron is connected to all neurons in the previous layer corresponding to points p' and index $i \in \{1..n\}$ using the weights

$$w^{EL}_{i,\text{type}(p),\text{type}(p'),\text{connection}(c,p,p')}$$

for both colours c.

3. Learning

The learning is described according to the usual distinction between training (subsection 3.1) and testing (subsection 3.2).

3.1 Training

Games for training are produced by self-play. A move is selected by using 1-ply look-ahead with the sum of all outputs in the evaluation layer as the scoring function.

Although training on larger board sizes provides more reinforcement signal for each position, the 1-ply look-ahead would slow it down considerably. Therefore the experiments were done on a 9×9 board. However, the network architecture allows retraining the network on increasing board sizes to adapt it to the different ratios between edge and centre points.

For better exploration of the state space, in 15% of the moves, instead of playing the move with the highest score, Gibbs sampling (Geman and Geman, 1984) over the move scores was used. The (unnormalised) probability of selecting a move with score s was

$$P(s) = \exp(s/T)$$

with a temperature T of 4.0. These positions were not trained.

After each played game, the 10 most recent games were trained using temporal-difference learning with $\lambda = 0$ (Sutton, 1988). The games were trained in random order going backward from the end position with immediate update of the weights after each position. The reason for the small value of λ is that most parts of the network see only a portion of the board, so that the effective length of the game is not the number of moves in the global game, but is the number of moves in a part of the board.

The weights were updated by backpropagation. All neurons in layers that receive a reinforcement signal by the temporal difference algorithm were treated as output neurons in the backpropagation algorithm. The algorithmically computed connections to and from the global connectivity layer did not take part in the backpropagation algorithm.

3.2 Testing

After the first 100 games and every 5,000 games thereafter, the performance was tested by playing 100 games on a 9×9 board against the program GNUGO version 3.0.0, released in 2001 (GNUGO, 2001). On the NNGS Go Server, the rating of GNUGO in 2001 was about 13 kyu (NNGS, 2001).

To obtain a variety of different games, every move of the network was selected by Gibbs sampling over the score with a temperature of 0.33. GNUGO always played White, the komi was 5.5. Identical games or games that could be mapped to other games by rotation and mirroring were sorted out.

The error of the mean value of the average score and percentage of wins is given by the standard deviation of the values divided by the square root of the number of games. However, this does not take into account partial correlations between the games. To get a more robust estimation of the error it is helpful to look at the deviation of the values late in the training process. At this time the changes in the weights of network are small, so that no big change in the playing strength is expected. From the reproducibility of the values between slightly different networks the error of the average score is estimated to be ± 5 points and the error of the percentage of wins $\pm 10\%$.

4. Experiments

The description of the experiments consists of two parts; the setup (subsection 4.1) and the results (subsection 4.2).

4.1 Setup

The size of the network was chosen to be 8 neurons per point in the first hidden layer and 2 neurons per point in the second hidden layer. The learning rate for the weight update was $3 \cdot 10^{-4}$. The performance of the network was compared using two kinds of input.

Raw board: Only 1 neuron per point was used in the input layer with constant activation 1. This corresponds to providing the network only with the raw Go position as input, because the location of the stones is already used implicitly in the selection of the weights from the weight sets according to the point types.

Preprocessed board: The position was preprocessed and some local features of the position were used as input for the network. Only simple features that can be computed quickly and non-expensive tactical searches were used. The features included: number of stones and liberties of blocks, a weighted sum of higher-grade liberties (PON-estimation for blocks without any concept of groups (Tajima and Sanechika, 1998)) and the results of simple tactical searches (ladders (Sensei, 2003)). Also, basic

link patterns (straight 2 and 3 point jump, knight jump, long knight jump) were detected. See Table 1 for a detailed listing of the inputs.

Input for empty points	
0...5	Black has 0, 1, 2, 3, 4, >4 liberties if playing here
6...11	White has 0, 1, 2, 3, 4, >4 liberties if playing here
12	Black can be captured in a ladder if playing here
13	White can be captured in a ladder if playing here
14	Single-point eye for Black
15	1 move necessary for single-point eye for Black
16	2 moves necessary for single-point eye for Black
17	>2 moves necessary for single-point eye for Black
18	Ponnuki shape for Black (Sensei, 2003)
19	1 move necessary for single-point eye for White
20	2 moves necessary for single-point eye for White
21	>2 moves necessary for single-point eye for White
22	Ponnuki shape for White
23	Move by Black here puts some white block in atari
24	Point is part of link pattern for Black
25	Point is part of link pattern for White
Input for occupied points	
0...7	PON is <-1.5, -0.5, 0.5, 1.5, 2.5, 3.5, 4.5, >4.5 (Tajima and Sanechika, 1998)
8	Block can be captured in a ladder if opponent moves first
9	Block can be captured in a ladder if its colour moves first
10...13	Number of liberties of block is 1, 2, 3, >3
14...18	Number of stones of block is 1, 2, 3, 4, >4

Table 1. Preprocessed input.

4.2 Results

The training took several weeks of CPU time on an Athlon XP 1800. Figure 4 shows the results of the test games against GNUGO. The network using the raw board input achieves an average score of about –25 points after 40,000 games. The network using the preprocessed input achieves an average score of about –5 points after 10,000 games.

Figures 5 and 6 show an example position with the evaluation output and the connectivity map for a point of the network using the preprocessed input. The network considers the left white group to be safe (0.2 is equivalent to 80% probability to become alive) but the centre group at F4 is unsafe (40% probability to become alive). The reason can be seen in the connectivity map for F4 in Figure 6: The probability for White to connect F4 to B4 is only 40%.

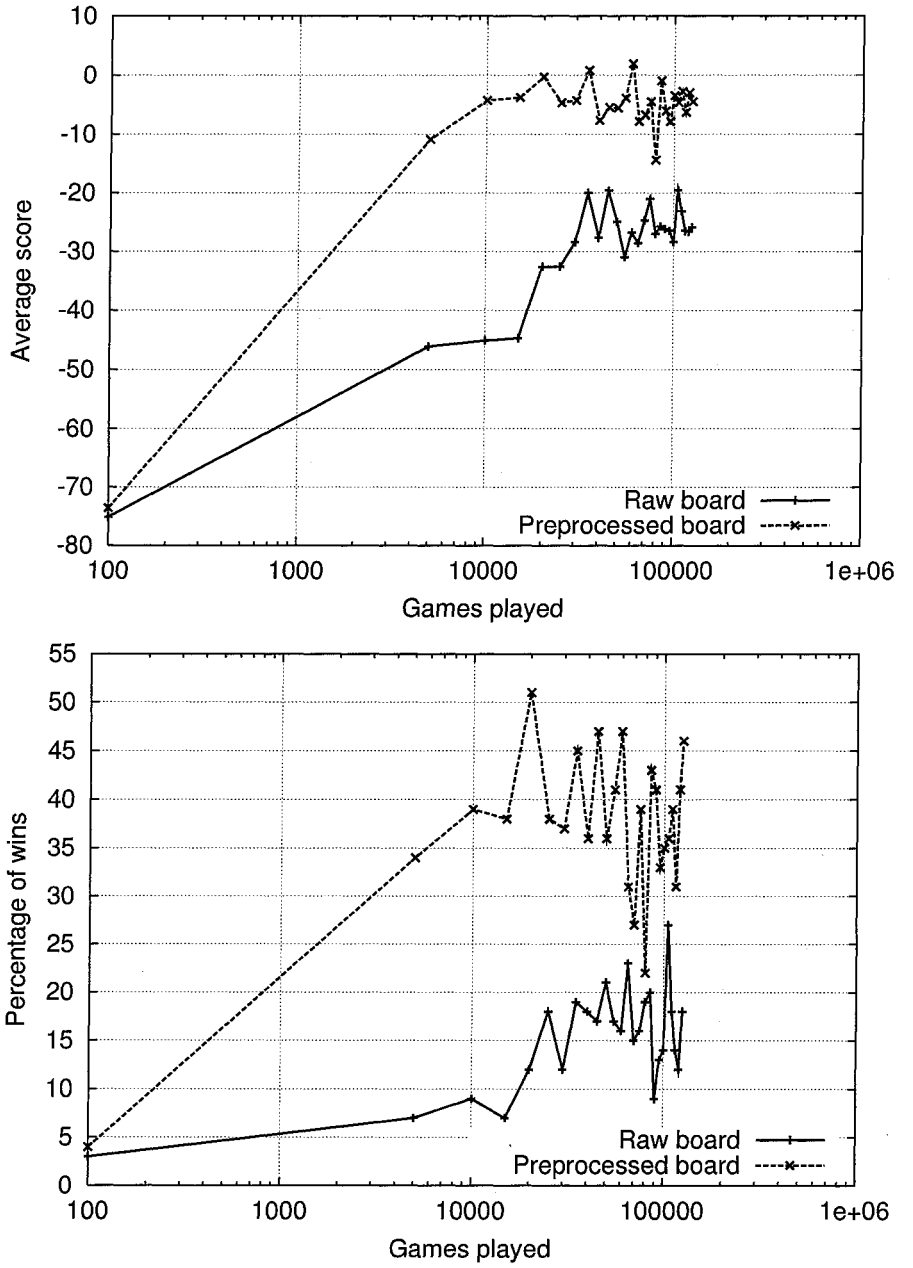


Figure 4. Average score and wins against GNUGO. The error of the average score is estimated to be ±5 points and the error of the percentage of wins ±10% (see subsection 3.2).

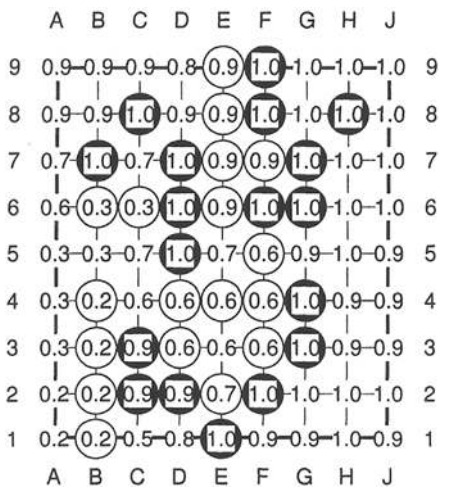


Figure 5. Example position (last move White E2). The numbers show the evaluation output of the network using preprocessed input.

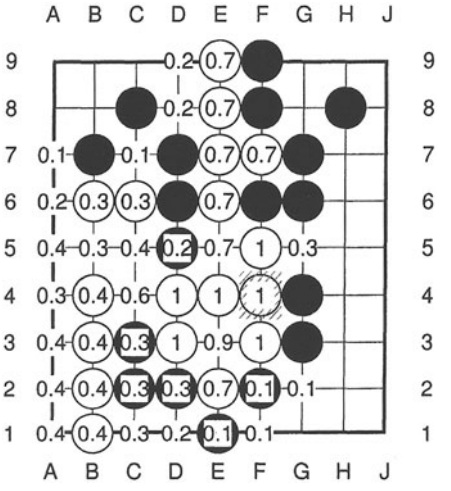


Figure 6. Example position (last move White E2). The numbers show the connectivity map of the network using preprocessed input for White from the point F4.

A complete game of the network versus GNUGO is shown in Figure 7. GNUGO played Black in this game. The game was played with the network using preprocessed input after the training was finished. The network does a good job in keeping the black stones separated (with one mistake at move White 36) and wins by 8.5 points.

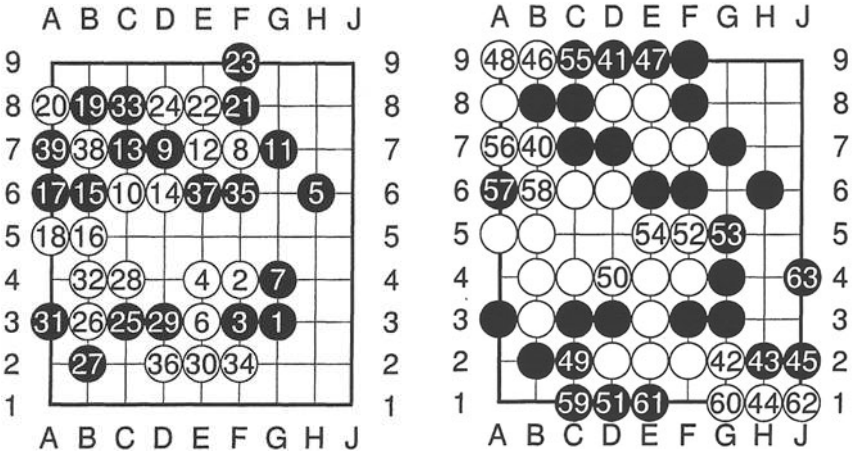


Figure 7. Example game of the network using preprocessed input versus GNUGO (here playing Black). White wins by 8.5 points.

5. Conclusion

It was shown that the presented neural-network architecture can be successfully used for evaluating Go positions. Considering that the best Go programs currently play at a level around 8 kyu, the good performance against a 13 kyu program is promising. In particular, the approach addresses a weakness that current Go programs have in handling complicated tactical situations with many nearby weak groups. However, it is clear that a static evaluation cannot handle all kinds of positions. Thus, it will be necessary to add more local tactical search results to the input, and/or use the network as an evaluation function in a global search.

The most current version of NEUROGO uses the described architecture with more neurons in the hidden layers and more sophisticated input features. This increases the average score against GNUGO 3.0.0 to about +2 points and the percentage of wins to about 50%.

References

Bouzy, B. and Cazenave, T. (2001). Computer Go: an AI oriented survey. *Artificial Intelligence*, 132(1):39–103.

Chen, Z. (2002). Semi-empirical quantitative theory of Go. *ICGA Journal*, 25(4):211–218.

Enzenberger, M. (1996). The integration of a priori knowledge into a Go playing neural network. http://www.markus-enzenberger.de/neurogo.html.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741.

GNUGO (2001). GnuGo Go program. http://www.gnu.org/software/gnugo/.

Müller, M. and Gasser, R. (1996). Experiments in computer Go endgames. In Nowakowski, R., editor, *Games of No Chance*, volume 29 of *MSRI Publications*, pages 273–284. Cambridge University Press, New York, NY.

NNGS (2001). No name Go server. http://nngs.cosmic.org.

Reitman, J. (1976). Skilled perception in Go. *Cognitive Psychology*, 8:336–356.

Schaeffer, J. (2001). A gamut of games. *AI Magazine*, 22(3):29–46.

Schraudolph, N. N., Dayan, P., and Sejnowski, T. J. (1994). Temporal difference learning of position evaluation in the game of Go. In *Advances in Neural Information Processing 6*. Morgan Kaufmann, San Francisco, CA.

Sensei (2003). A glossary of go terms. http://senseis.xmp.net/?GoTerms.

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning*, 3:9–44.

Tajima, M. and Sanechika, N. (1998). Estimating the possible omission number for groups in Go by the number of n-th dame. In van den Herik, H. J. and Iida, H., editors, *Computers and Games, First International Conference*, volume 1558 of *Lecture Notes in Computer Science*, pages 265–281. Springer-Verlag, Berlin, Germany.