Learning to Predict Life and Death from Go Game Records

Erik C.D. van der Werf, Mark H.M. Winands, H. Jaap van den Herik and Jos W.H.M. Uiterwijk *
Institute for Knowledge and Agent Technology, Department of Computer Science, Universiteit Maastricht,
P.O. Box 616, 6200 MD Maastricht, The Netherlands.

Abstract

This paper presents a learning system for predicting life and death in the game of Go. Learning examples are extracted from game records. On average our system correctly predicts life and death for 88% of all blocks. Towards the end of a game the performance increases up to 99%. Clearly, such a predictor will be an important component for building a full-board evaluation function.

1 Introduction

Evaluating Go positions is one of the hardest tasks in Artificial Intelligence (AI). In the last decade Go has received significant attention from AI research [1, 9]. Yet, despite all efforts, the best computer Go programs are still weak compared to human players. Partially this is due to the complexity of 19×19 Go. However, even on the 9×9 board, which has a complexity between Chess and Othello [1], the *current* Go programs perform nearly as bad. The main reason lies in the lack of good positional evaluation functions. Many (if not all) of the current top programs rely on (huge) static knowledge bases conceived by the programmers as they understand the Go skills and Go knowledge of a Go player. As a consequence the top programs are extremely complex and difficult to improve. In principle a learning system should be able to overcome this problem.

Over centuries humans have acquired extensive knowledge of Go. Since that knowledge is implicitly available in the games of human experts, it should be possible to apply machine-learning techniques to extract that knowledge from game records. One of the best sources of game records on the Internet is the NNGS archive [10]. Although the NNGS game records contain a wealth of information, the automated extraction of knowledge from these games is a non-trivial task at least because of incomplete scoring information, unfinished games, and bad moves during the game.

Recently, we have set the first step towards making the knowledge in the game records accessible. We have built a learning system that is able to score automatically 98.9% of the final positions correct without any human intervention [12]. The reliable scores were obtained by a highly accurate classification of life and death for final positions ($\sim$99.7% of all blocks correct). As a result we now have a database containing 18,222 9×9 games with reliable and complete score information. From this database we intend to learn relevant Go skills which enable us to build a strong positional evaluation function. In this paper we present our latest work towards this end, which focusses on learning to predict life and death during the game. We believe that predicting life and death is a skill which is pivotal for strong play and an essential ingredient in any strong positional evaluation function.

The rest of this paper is organised as follows: Section 2 presents the learning task. Section 3 introduces the representation. Section 4 provides details about the dataset. Sections 5 and 6 report our experiments. Finally, Section 7 presents our conclusions.

2 The learning task

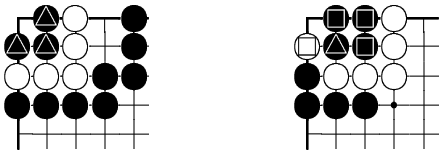
Predicting life and death can be learned from a set of labelled game records, for which the labels contain the colour controlling each point at the end of the game. An intuitively straightforward implementation would classify each block as the (majority of) occupied labelled points. Unfortunately this does not necessarily provide correct information for classification of life or death, for which at least two conflicting definitions exist.

The Japanese Go rules state: "Stones are said to be 'alive' if they cannot be captured by the opponent, or if capturing them would enable a new stone to be played that the opponent could not capture. Stones which are not alive are said to be 'dead'."

The Chinese Go rules state: "At the end of the game, stones which both players agree could inevitably be captured are dead. Stones that cannot be captured are alive."

A consequence of both rules is shown in Figure 1a: the marked black stones are considered alive by the Japanese rules, and dead by the Chinese rules. Since the white stones are dead under all rule sets, scoring such a

* Email:{e.vanderwerf,m.winands,herik,uiterwijk}@cs.unimaas.nl



(a) (b)

Figure 1: Alive or dead?

position is not a problem. However, whether the black stones should be considered alive or dead is unclear.

Another problematic position is shown in Figure 1b. If this position is scored under the Japanese rules all marked stones are alive. However, if the position would be played out the most likely result (which may be different if one side can win a ko-fight) is that the empty point in the corner, the marked white stone and the black stone marked with a triangle become black, and the three black stones marked with a square become white. Furthermore, all marked stones are captured and therefore dead under the Chinese rules.

In this paper we choose the Chinese rules for defining life and death. The learning task therefore becomes the task of predicting whether blocks of stones can be captured. When replaying the game backwards the following four classes of blocks can be identified (in order of decreasing domination):

1. Blocks that are captured during the game.
2. Blocks that occupy points ultimately controlled by the opponent.
3. Blocks that occupy points on the edge of regions ultimately controlled by their own colour.
4. Blocks that occupy points in the interior of regions ultimately controlled by their own colour.

Blocks of class 1 and 2 should be classified as dead. Blocks of class 3 should be classified as alive. Class 4 blocks cannot be classified based on the labelling and are therefore not used in training.

Obviously, perfect classification is not possible in non-final positions. Therefore the goal is to approximate the Bayesian *a posteriori* probability or at least the Bayesian discriminant function, for deciding whether the block will be alive or dead at the end of the game. In pattern recognition there are several ways to do this. A popular choice is the use of a multi-layer perceptron (MLP) classifier. It has been shown [6] that minimising the mean-square error (MSE) on binary targets, for an MLP with sufficient functional capacity, adequately approximates the Bayesian *a posteriori* probability.

3 Representation of the block

Many representations for characterising blocks are possible and used in the Computer-Go field. The most primitive representations typically employ the raw board directly. Although such representations are complete, in the sense of containing all relevant information, they are known to be inefficient because of their high dimensionality and lack of topological structure. Our representation employs features based on simple measurable geometric properties, some elementary Go knowledge and some hand-crafted specialised features. Several of our features are typically used in Go programs to evaluate positions [2, 4]. The features are calculated for single friendly and opponent blocks, multiple blocks in chains, and colour-enclosed regions (CERs).

For each block we include the following features: *Size, Perimeter, Adjacent opponent stones, Number of first-, second-, and third-order liberties, Protected liberties, Auto-atari liberties, Adjacent opponent blocks, Local majority, Centre of mass*, and *Bounding box*.

Adjacent to each block are CERs consisting of connected empty and occupied points, surrounded by stones of one colour or the edge. It is important to know whether an adjacent CER is fully accessible, because a fully accessible CER provides at least one sure liberty. To detect fully accessible regions we use so-called miai strategies as applied by Müller [8]. In contrast to Müller's original implementation we also add miai accessible interior empty points to the set of accessible liberties, and also use protected liberties for the chaining.

For fully accessible CERs we include: *Number of regions, Size, Perimeter*, and *Split points*.

For partially accessible CERs we include: *Number of regions, Accessible size, Accessible perimeter, Unaccessible size, Unaccessible perimeter*, and *Unaccessible split points*.

The size, perimeter, and number of split points are summed for all regions. We do not address individual regions because the representation must have a fixed length, whereas the number of regions is not fixed.

Another way to analyse CERs is to look for possible eyespace. Points forming the eyespace should be empty or contain capturable opponent stones. Empty points directly adjacent to opponent stones are not part of the eyespace. Points on the edge with one or more diagonally adjacent opponent stones and points with two or more diagonally adjacent opponent stones are false eyes. False eyes are not part of the eyespace (we ignore the unlikely case where a big loop upgrades false eyes to true eyes). For directly adjacent eyespace of the block we include: *Size* and *Perimeter*.

In many positions blocks of the same colour with shared liberties will be connected at the end of the game. An optimistic scenario therefore may assume that all blocks with shared liberties can form a chain. For this, so-called, optimistic chain we include: *Number of blocks, Size, Perimeter, Split points, Adjacent CERs, Adjacent CERs with eyespace, Fully accessible adjacent CERs, Size of adjacent eyespace, Perimeter of adjacent eyespace*, and *External opponent liberties*.

Adjacent to the block in question there may be opponent blocks. For the weakest (measured by the number of liberties) directly adjacent opponent block we include: *Perimeter, Liberties, Shared liberties, Split points*, and *Perimeter of adjacent eyespace*. The same features are also included for the second-weakest directly adjacent opponent block, and the weakest opponent block directly adjacent to or sharing liberties with the optimistic chain of the block in question.

By comparing a flood fill starting from Black with a flood fill starting from White we can find unsettled empty regions which are disputed territory. If the block is adjacent to disputed territory we include: *Direct liberties, Liberties of all friendly blocks*, and *Liberties of all enemy blocks* in the disputed territory.

Finally, we include the following global features: *Player to move, Ko, Distance to ko, Number of friendly stones*, and *Number of opponent stones*.

4 The data set

In the experiments we use 9×9 game records played between 1995 and 2002 on NNGS [10]. We only use games which are played to the end and scored. Since NNGS game records only contain a single numeric value for the score, the fate of all points had to be labelled by an external program. For this we used GNUGO [5], some manual labelling, and our own learning system for scoring final positions [12]. Since automatic labelling is still imperfect, all games where the score based on the labelling was not identical to the numeric score in the game record, or where the final positions contained unsettled interior points, were inspected manually. Finally, an additional inspection of a few hundred randomly selected final positions revealed that none were labelled incorrectly.

In all experiments test examples were extracted from games played in 1995, training examples were extracted from games played between 1996 and 2002. Since the games provide a huge amount of blocks with little or no variation (large regions remain unchanged per move) only a small fraction of blocks were randomly selected for training ($< 5\%$ per game).

5 Choosing a classifier

An important choice is selecting a good classifier. Our previous work on scoring final positions showed that the MLP classifier provides good performance with a reasonable training time [12]. The performance of the MLP mainly depends on the architecture, the number of training examples, and the training algorithm. In the experiments reported here we tested architectures with 1 and with 2 hidden layers containing various numbers of neurons per hidden layer. For training we compared: (1) gradient descent with momentum and adaptive learning (GDXNC) with (2) RPROP backpropagation (RPNC) [11]. For comparison we also present results obtained for the Nearest Mean Classifier (NMC), the Linear Discriminant Classifier (LDC), and the Logistic Linear Classifier (LOGLC). More information on these classifiers can be found in [3] and [7].

Classifier	Test error (%)		
Training examples	1000	5000	25000
NMC	21.5	21.0	21.0
LDC	14.2	13.7	13.6
LOGLC	14.8	13.3	13.1
GDXNC-5	13.8	12.9	12.2
GDXNC-15	13.9	12.9	12.2
GDXNC-25	13.7	12.8	12.0
GDXNC-25-5	13.8	12.9	12.1
GDXNC-25-25	13.9	12.8	12.0
GDXNC-50	13.8	12.8	12.0
RPNC-5	14.7	13.4	12.4
RPNC-15	14.4	13.2	12.4
RPNC-25	14.7	13.3	12.4
RPNC-25-5	14.3	13.4	12.6
RPNC-25-25	14.3	13.5	12.8
RPNC-50	15.0	13.3	12.5

Table 1: Performance of classifiers. The numbers in the names indicate the number of neurons per hidden layer.

The results, shown in table 1, indicate that GDXNC performs slightly better than RPNC. Although RPNC using RPROP provides faster training ($2\sim3$ times faster than GDXNC) and a lower error on the training data, the performance on the test data seems to be worse because of overfitting. Using one hidden layer with 25 neurons is sufficient at least for training with 25,000 examples. Adding a second hidden layer with 5 or 25 neurons does not improve performance.

6 Performance over the game

In the previous section we calculated the average classification performance over whole games. Although this is an interesting measure, it does not tell us how the performance changes as the game develops. We believe that for standard opening moves the best choice is pure guessing based on the highest *a priori* probability (always alive). Final positions, however, can (at least in principle) be classified perfectly. Given these extremes it is interesting to see how the performance of our system changes over the game, either looking forward from the beginning or backward from the end.

Figure 2a shows that pure guessing performs equal for roughly the first 10 moves. As the length of games increases the *a priori* probability of blocks on the board ultimately being captured also increases (which makes sense because the best points are occupied first and there is only limited space on the board).

Good evaluation functions aim at predicting the final result of the game as soon as possible. It is therefore encouraging to see in Figure 2b that towards the end the error goes down rapidly, predicting about 95% correct when 10 moves before the end of the game. For final positions our system classifies over 99% of all blocks correctly. Our previous work [12] showed that this performance is at least comparable to that of the average rated NNGS player. Whether this performance is similar for non-final positions is difficult to say.

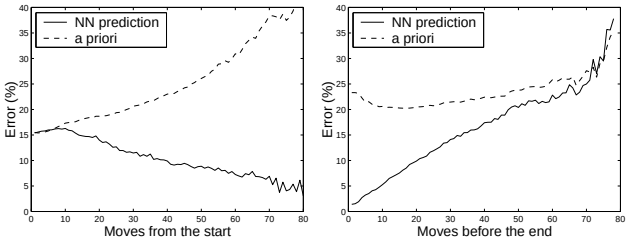
7 Conclusions and future research

We have developed a system that learns to predict life and death from labelled examples. On unseen game records our system, averaged over the whole game, classifies around 88% of all blocks correctly. Ten moves before the end of the game it predicts around 95% correct, and for final positions over 99% are classified correctly.

To obtain more insight into the importance of this work, life-and-death prediction will be incorporated in a full-board evaluation function for predicting the outcome of games. Testing the full-board evaluation function on non-final positions and resigned games (which were not played to completion) will be an important measure.

Acknowledgements

We gratefully acknowledge financial support by the Universiteitsfonds Limburg / SWOL.



(a) (b)

Figure 2: Performance over the game

8 References

[1] B. Bouzy and T. Cazenave. Computer Go: An AI oriented survey. *Artificial Intelligence*, 132(1):39–102, October 2001.

[2] K. Chen and Z. Chen. Static analysis of life and death in the game of Go. *Information Sciences*, 121:113–134, 1999.

[3] R.P.W. Duin. *PRTools, a Matlab Toolbox for Pattern Recognition*, 2000.

[4] D. Fotland. Static eye analysis in 'THE MANY FACES OF GO'. *ICGA Journal*, 25(4):201–210, 2002.

[5] GNUGO, 2003. http://www.gnu.org/software/gnugo/.

[6] J. B. Hampshire II and B. A. Perlmutter. Equivalence proofs for multilayer perceptron classifiers and the Bayesian discriminant function. In *Proceedings of the 1990 Connectionist Models Summer School, 1990. D. Touretzky, J. Elman, T. Sejnowski, and G. Hinton, eds. Morgan Kaufmann, San Mateo, CA.*, 1990.

[7] A. K. Jain, R. P. W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[8] M. Müller. Playing it safe: Recognizing secure territories in computer Go by using static rules and search. In H. Matsubara, editor, *Proceedings of the Game Programming Workshop in Japan '97*, pages 80–86. Computer Shogi Association, Tokyo, Japan, 1997.

[9] M. Müller. Computer Go. *Artificial Intelligence*, 134(1-2):145–179, January 2002.

[10] NNGS. The no name go server game archive, 2002. http://nngs.cosmic.org/gamesearch.html.

[11] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation: the RPROP algorithm. In *Proceedings of the IEEE Int. Conf. on Neural Networks (ICNN)*, pages 586–591, 1993.

[12] E. C. D. van der Werf, H. J. van den Herik, and J. W. H. M. Uiterwijk. Learning to score final positions in the game of Go, 2003. Submitted for publication.