Proceeding of the International Conference on Computing, Communications and Control Technologies,
August 14-17, 2004, Austin, Texas, USA

Learning to Predict Life and Death with ANN in Computer Go

Julian WONG, Wai Hung & Andy CHUN, Hon Wai
Department of Computer Science
City University of Hong Kong
Tat Chee Avenue, Kowloon
Hong Kong SAR
50249864@plink.cityu.edu.hk & *andy.chun@cityu.edu.hk*

ABSTRACT

Tsumego (死活– Life and Death) is a computer Go game sub-problem that determines whether a group of stones is safe and free from ever being captured – the "life" state, or may potentially be captured after a sequence of moves, no matter how he/she defenses it – the "death" state. The ability to quickly and accurately determine "life and death" is crucial to becoming an expert at Go. Humans usually learn this skill by studying examples in classic Go books written by professional players. Most examples usually use only a corner of the Go board or a side of the board to illustrate how to archive "alive" or "dead" state. This paper shows how we designed and used an Artificial Neural Network (ANN) to simulate this learning process. We performed two different experiments – (1) stones in "final position" and (2) stones not in "final position. Using back propagation learning algorithm with a training set of over a 1000 examples of 8x8 corner positions, we were able to achieve a very high accuracy rate of 97% for final position tests and 94.5% for non-final position tests. This paper describes the design of our ANN and algorithm as well as provides an analysis and comparison of our experiment results.

Keywords: Computer Go, Artificial Neural Network, Learning.

1. INTRODUCTION

Although not too many research projects dealt with the Life and Death problem, we were able to find a few to compare our work with. For example, Thomas Wolf created a program call GoTools [3] which was specifically designed to solve Life and Death. The GNU Go [4] Documentation also has some explanation on how GNU Go solves Life and Death. Horace Chan's M.Phil Thesis [5] also performed related ANN research on Life and Death. The following compares our approach with these three approaches.

Most of the publications related to Life and Death were published by Thomas Wolf, the creator of GoTools [3], a proprietary software that solves Life and Death problems. From [6, 7], Wolf revealed that GoTools used a heuristic approach. In the homepage of GoTools [3], it states that GoTools has been ranked as 5-dan by a Nihon Ki-in (Japan Go Association) ranked amateur 6-dan player, in solving Life and Death. 5-dan is a fairly high ranking. The ranking system in Go starts at 30 kyu (weakest) and moves up to 1 kyu. After 1 kyu (1k) is 1 dan, with 7 dan the highest ranking.

According to IntelligentGo.org [8], it claimed that GoTools can actually solve higher dan-level tsumego problems. However, we were not able to find the actual data or performance statistics.

We tested our benchmark cases with the free Java applet version of GoTools. We tested two 1k, one 4k and five 5k problems take from a tsumego book written by the Japanese professional player Akinobu Chobi [9]. All these problems were final Life and Death patterns. Out of the eight problems, GoTools unfortunately only solved of the 5k problems correctly. All others turn out to be miss or timeout (300 sec. time out). When we selected the medium speed search, it got the two questions correct, one time out and all other missed. For the other two slower search methods, it returned a wrong answer for one of the patterns and timed out on all others. However this is a test on the Java applet version of GoTools and might not directly reflect the real strength of the actual program.

On the other hand, GNUGo [4] uses one module to extract the Life and Death patterns and passes it to another module for evaluation. The pattern reading program is called OWL (Optics with Limit Negotiation). OWL depends on three pattern databases, which expend and limit the moves in the OWL reading. Two different modules can be used to evaluate the positions from OWL. One is "optics" and the other is "life". Both perform the same job but "life" is more accurate and slower than "optics". However we cannot found any data regarding the official ranking of the Life and Death reading modules. For the overall strength of GNU Go version 3.2, it was ranked at 6k* from the Legend Go Server (LGS) [16].

LGS is an Internet Go Server which will estimate a player's rank when they play with other ranked players for more than 15 games. Every player can set her/his strength from 30k – 7d. After the system estimates the player's strength, there will be a '*' after the rank. For example, GNU Go Version 3.2 is ranked 6k* by LGS.

To the best of our knowledge, the only research related to using ANN on Life and Death is from Horace Chan's M.Phil. thesis at the Chinese University of Hong Kong [5]. He used a specially designed ANN to recognize final-position life and death patterns and obtained a 96.9% accuracy rate from 387 test patterns of human-selected training data set. Our first experiment is similar to his experiments. However, we went beyond just evaluating final position and conducted a second experiment with non-final patterns. Our ANN design is somewhat different from Chan's in structure; it is smaller. Our experiment results also show that our ANN model seems to produce more accurate results.

2. OUR EXPERIMENTS

For ANN training and testing, game samples were taken from amateur games as well as classic Go texts. The amateur games were downloaded from the Legend Go Server [16]. The final score requires both sides to agree on the life status of every group on the board. However, weaker players might not be able to determine life status accurately, so only amateur games at the dan level with final scores will be used for our experiments. Our experiments use inputs within a corner smaller than 8x8. "Seki" and "Ko" situations [4] are not included as samples are too hard to collect ("seki" are special cases where both sides need to sacrifice and "ko" are repeated moves than never end). When a pattern exceeds the 8x8 corner size, it will be trimmed manually if possible while not affecting the outcome of life and death.

Experiment 1 – Final Position
In the first experiment, we trained our ANN to determine life/death status of stones in final position; resigned games were not used. In final game position, the score and life/death status for each group on board would have been agreed upon by both players. Therefore, all alive/dead stones are cleared specified. We then extract those residing in the 8x8 corner for our experiments. Data used in Experiment 1 is similar to the human-selected training data set of Chan [5].

Experiment 2 – Not in Final Position
In the second experiment, new sample and test cases were added to the data pool used in the first experiment. Life and death problems were extracted from book written by professional players [9, 10]. This includes stone patterns that are not in final position. The life and death problems that we used were below the 3-kyu level. All dead patterns are considered final positions, since no matter what move one makes, the opponent will always be able to counter it, unless the opponent makes a mistake.

There are significant differences in stone patterns not in final positions compared with those in final positions. Because of the differences, even if a "life and death" program was successful in recognizing final positions, it might not perform at all well during the actual game. Hence we believe Experiment 2 is important in measuring the performance of "life and death" software.

3. OUR ANN DESIGN

We used the FANN C-library [11] to construct our back-propagation ANN. Two different ANN structures were tested on both Experiments 1 and 2.

Fully Connected ANN
We used a 64-64-30-2 and 64-64-30-1 network for the first and second experiment respectively, i.e. 64 input nodes, 64 nodes in the first hidden layer and then 30 in the second hidden layer. Both networks were fully connected. The 64-64-30-2 network used threshold function at the output layer where "1 0" means alive and "0 1" means dead. The 64-64-30-1 network, on the other hand, used a sigmoidal function where "0" means dead and "1" means alive. Configuration like 64-128-50-2, 64-128-50-20-2, and 64-128-50-30-2 were also tested. It was found that increasing the network size might not necessarily improve the accuracy rate while taking much longer to train. We followed the general delta rule on changing the link weights. Learning rate with 0.5 and 0.7 were used and the results were found to be similar.

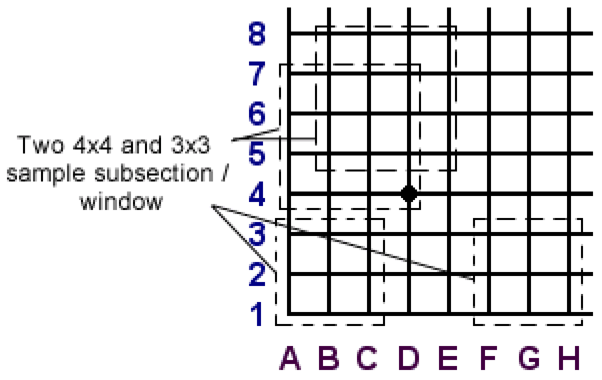

Fig. 1. The input to our partially connected ANN

Partially Connected ANN

Beside the fully connected ANN, a 64-140-50-30-2 and a 64-140-50-30-1 network with specially designed partial connections between the input and first hidden layer were also used in experiments one and two respectively. The board was first divided into subsections (see Fig. 1). Each neuron in the first hidden layer only connects to a particular subsection of input neuron. For example, the 9 neurons representing the inputs A1, A2, A3, B1, B2, B3, C1, C2, and C3 formed a 3x3 subsection. One of the neurons in first hidden layer will connect to these nine neurons. We divided the board into 49 2x2 subsections, 36 3x3 subsections and so on. Therefore, in total there are 49+36+25+16+9+4+1 = 140 neurons in the first hidden layer with each represent a single subsection. Fig. 2 illustrates our partial connected ANN design.

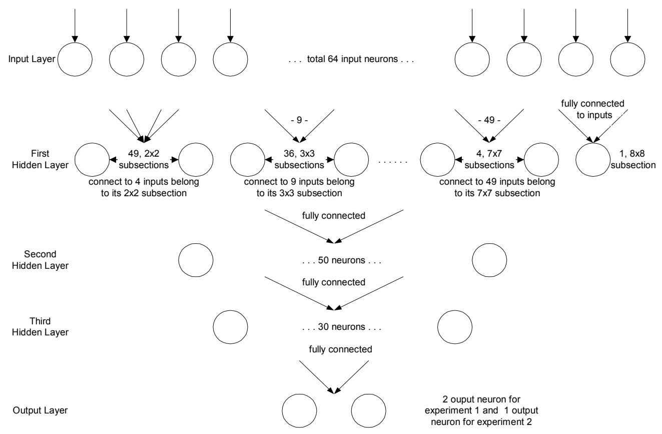
Input Layer
. . . total 64 input neurons . . .
First Hidden Layer
49, 2x2 subsections
connect to 4 inputs belong to its 2x2 subsection
- 9 -
36, 3x3 subsections
connect to 9 inputs belong to its 3x3 subsection
- 49 -
4, 7x7 subsections
connect to 49 inputs belong to its 7x7 subsection
fully connected to inputs
1, 8x8 subsection
fully connected
Second Hidden Layer
. . . 50 neurons . . .
fully connected
Third Hidden Layer
. . . 30 neurons . . .
fully connected
Output Layer
2 ouput neuron for experiment 1 and 1 output neuron for experiment 2

Fig. 2. The structure of the partially connected ANN

By pre-encoding some domain knowledge of two-dimensional proximity into the ANN structure, we believe the network will perform better. A similar idea was used for Checkers [12] and in Chan's experiments [5].

For Chan's design, he used 128 neurons and 150 neurons in the input and first hidden layer respectively. He also used a similar subsection design [5] as we did. However, he started at 3x3 instead of 2x2 like we did.

4. OUR PROGRAMS

We first downloaded test/learning board patterns from Go Game servers in SGF format [17], a commonly used format to store Go games. Additional patterns were obtained from classic Go texts, also manually coded in SGF format.

Perl scripts were then used to post-process the SGF files. The scripts were used for several tasks. Firstly, the 8x8 corner of interest need to be rotated to the top left corner. Secondly, we need all the test/training cases to use white stones as the enclosing stones and black for the enclosed stones. Thirdly is the isomorphic patterns generation (see next section).

After this processing, the data files were then separated into a file for training samples and another for the test cases. The training data was then feed to our ANN, which we coded in C using the FANN library [11].

5. RESULTS

We used 748 patterns for Experiment 1. It contained 298 dead patterns from Go books and 450 "life and death" patterns from final game positions. We randomly selected 200 out from the pool as test cases. Then the remaining 548 patterns were "flipped" to obtain another 548 isomorphic patterns for training. Figures 3 and 4 illustrate an example of an isomorphic pattern pair.

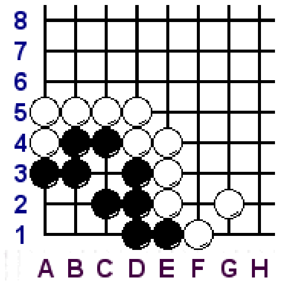

Fig. 3

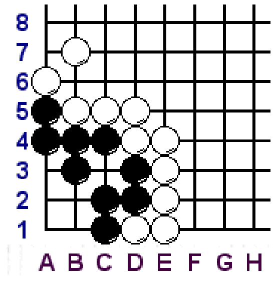

Fig. 4

In Experiment 2, 252 alive patterns not in final position were added. This time we randomly select 250 out from the pool to be test cases. For Experiment 2, the ANN need to recognize 3 types of patterns instead of 2 – alive (final), dead, and alive (not final). There is 50 more test cases compared with Experiment 1 as the ANN need to classify an addition type of pattern. The additional tests make sure the ANN's recognition rate is a more accurate measurement. Also, similar to Experiment 1, isomorphic patterns of the remaining samples were used to generate additional cases for training.

For our experiments, we used an approximation that if $(\text{EXPECTED-OUTPUT} - \text{ANN-OUTPUT})^2 < (0.25)^2 = 0.0625$, we count it as correct. EXPECTED-OUTPUT will be 1 for alive patterns (no matter it is in final position or not) and 0 for dead patterns.

Tests:	**Experiment 1 Accuracy**	**Experiment 2 Accuracy**
64-64-30-2/1 fully connected ANN	94% 188 / 200	89.6% 224/250
64-140-50-30-2/1 partially connected ANN	97% 196 / 200	94.8% 237/250
Chan [5] – human selected training data set	96.9%	Nil
Chan [5] – mixed training data set	98.45%	Nil

For the Experiment 1, we obtained a 97% accuracy rate, which was almost identical to results obtained by Chan [5], which was 96.9%. Chan further improved his results by hand-selecting training cases. Unfortunately, we do not have access to Chan's hand-selected data set, so we cannot make further comparisons.

On the other hand, we extended our research to not only cover games in final position, but also non-final positions. Performing "life and death" determination of stone patterns not in final position is a much more difficult problem but necessary to solve if the technique was to be used to support actual game playing. For this objective, we developed Experiment 2 and obtained a 94.8% accuracy rate. Unfortunately, we were not able to find any other research papers that also performed similar experiments to compare with. However, we believe the 94.8% accuracy rate is extremely high for this complex task.

6. CONCLUSIONS

In this paper we presented the design of our ANN structures to solve the "life and death" problem in Computer Go games. We compared two types of networks – fully connected and partially connected. We used these two types of networks to perform learning on two different data sets – patterns that were in final positions (Experiment 1) and patterns not in final positions (Experiment 2). In total, our ANN learned from over a thousand examples and then tested on over 200 new test cases. The recognition rate for Experiment 1 was 94% and 97% for fully-connected and partial-connected ANN respectively. The recognition rate for Experiment 2 was 89.6% and 94.8% for fully-connected and partial-connected ANN respectively.

The partially-connected ANN performed much better probably because of the domain knowledge of spatial proximity that was already encoded into the ANN structure.

We are encouraged by the results of these experiments to continue to explore how the ANN might be used during actual Computer Go game playing. The next step is to explore whether the ANN can, besides just recognition, be used to produce an actual sequence of moves for the a life and death problem. For example, by using a mini-max game tree, we can ask the ANN to generate a score for each board position. A board position generating function has already been created. We believe that even though the ANN by itself might not be able to accurately find the entire sequence of moves, if the ANN is accurate enough to just provide the first move that will greatly decrease the search space already and improve our chances of winning.

7. ACKNOWLEDGEMENT

The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. 9040517, CityU 1109/00E). The work described in this paper was also partially supported by a grant from the City University of Hong Kong (Project No. 7001286 and 6980002).

8. REFERENCES

[1] *Sensei's Library*, (n.d.) Retrieved 1 May 2004 from http://senseis.xmp.net

[2] Yen, S.J. and Shun-Chin Hsu, "A Positional Judgment System for Computer Go," *Advances in Computer Games*, Volume 9, Universiteit Maastricht, 2001, pp. 313-326.

[3] Wolf, T., (20 Nov 2001), *GoTools - the Tsume-Go program*, Retrieved 1 May 2004 from http://www.qmw.ac.uk/~ugah006/gotools/

[4] *GNU Go 3.4 Documentation* (3 Dec 2003), Retrieved 1 May 2004 from http://www.gnu.org/software/gnugo/gnugo_toc.html

[5] Chan, W.K. Horace, "Application of Temporal Difference Learning and Supervised Learning in the Game of Go" M.Phil Thesis, Chinese University of Hong Kong, 1996.

[6] Wolf, T., "The Program GoTools and its Computer-Generated Tsumego Database" In the *Proceedings of the Game Programming Workshop in Japan'94*, Hakone/Japan, Oct. 1994, pp. 84-96.

[7] Pratola, M. and Thomas Wolf, (2003) “Optimizing GoTools’ search heuristics using Genetic algorithms” *ICGA Journal*, Volume 26, 2003, No 1, pp. 28-49.

[8] *IntelligentGo.org Foundation* (n.d.), “Overview of Computer Go,” Retrieved 1 May 2004 from http://www.intelligentgo.org/en/computer-go/overview.html

[9] Akinobu Chobi (戸沢昭宣), “Go Game 9 kyu to 1 kyu,” (九級から一級までの詰碁), Seibido Shuppan (成美堂出版), Japan, ISBN: 4415044239, 1992.

[10] Hayashi (林海峯), translated by Ng (吳仁), “Fundamentals of Life and Death” (死活的基礎), Mercury Publishing House (世界文物出版社), Taiwan ROC, ISBN, 1982.

[11] Fast Artificial Neural Network Library (fann) (31 March 2004), Retrieved 1 May 2004 from http://fann.sourceforge.net

[12] Fogel, D., *Blondie 24*, Morgan Kaufmann, ISBN 1558607838, October 2001.

[13] Chellapilla, K. and David Fogel, “Evolving Expert Checkers playing Program without Using Human Expertise,” IEEE Trans. Evolutionary Computation, 5(4): 2001, pp. 422-428.

[14] Chellapilla, K. and David Fogel, “Evolution, Neural Networks, Games, and Intelligence” In the *Proceedings of the IEEE*, September, 1999, pp. 1471-1496.

[15] Chellapilla, K. and David Fogel, “Evolving Neural Networks to Play Checkers without Relying on Expert Knowledge” In the *IEEE Transactions on Neural Networks*, Volume 10, No. 6, November, 1999.

[16] *The Legend Go Server,* 傳奇圍棋網, (n.d.) Retrieved 1 May 2004 from http://lgs.taiwango.net

[17] *Hollosi, A.* (n.d.) SGF File Format FF[4] – Smart Game Format, Retrieved 1 May 2004 from http://www.red-bean.com/sgf/