

# A HYBRID ARTIFICIAL INTELLIGENCE APPROACH WITH APPLICATION TO GAMES

RICHARD CANT, JULIAN CHURCHILL, DAVID AL-DABASS

Department of Computing and Mathematics  
The Nottingham Trent University  
Nottingham NG1 4BU.  
Email: richard.cant/david.al-dabass@ntu.ac.uk

**Abstract:** We describe a hybrid Artificial Intelligence (AI) approach combining soft AI techniques (neural networks) and hard AI methods (alpha-beta game tree search), in an attempt to approximate human play more accurately, in particular with reference to the game of Go. The program is tested and analysed by play against another Go playing program and it is shown that the use of hard AI enhances the performance of the soft AI system and vice-versa.

**Keywords:** Neural Networks, Alpha beta search algorithms, Computer Go.

## INTRODUCTION

This paper investigates the combination of hard and soft Artificial Intelligence (AI) techniques and the application of such a combination to games. This paper concerns itself with applying the problem to the game of Go. Current Go playing programs have been markedly less successful than their chess playing counterparts. Whilst success in playing chess has come from a move away from attempting to copy human play, this approach has failed in the field of Go. In the present paper we explore an idea inspired by human modes of thought. We attempt to combine neural network techniques and traditional game tree search with some degree of success, allowing experienced based move selection to be used with rigorous analysis in an efficient and effective way.

### *What is Go?*

Go is a relatively simple game the complexity of which emerges as you become familiar with the ideas presented. A comparison with Chess is often made, as these are both board-based games of zero-chance [2]. The rules are simpler in Go, however the board is larger and due to the unrestrictive nature the rules there are many more moves available for the Go player to consider.

The game is played on a board, which has a grid of 19x19 intersections. Two players, black and white, take turns to place a single stone on any unoccupied intersection, with the aim of surrounding as much territory as possible. A player can pass at any turn (giving one point to his opponent) instead of placing a stone. Capturing the opponent's stones is

also used to increase a player's score. A stone is captured when the last of its liberties is removed. A liberty is an empty intersection directly next to the stone. Suicide is not allowed unless it is to capture some opponent's stones.

The end of the game is usually reached by mutual agreement between the players, when they both pass consecutively. Stones which are effectively dead and territory points are then totalled up and the winner declared.

## CURRENT RESEARCH

### *Neural Networks*

The inspiration behind the neural network idea is the simulation of neurons in a biological brain. Biological neurons receive stimulus signals from other neurons and when a certain activation level is reached the neuron fires signals to all the other connecting neurons. The change in the strength of the connections is dynamic and it is this particular feature that means networks of neurons simulated on a computer can be trained to recognise patterns of input and give appropriate patterns of output [3].

### *Hard And Soft AI*

For the purposes of this project it is necessary to define two ideas to allow us to discuss the hybrid approach used.

Hard AI refers to the more traditional artificial intelligence techniques such as the various tree search methods, pattern matching and rule based expert systems. The term 'hard' is used to emphasise the predictable and exact nature of such methods.

Soft AI techniques on the other hand deal with methods that rely on statistical processes and may produce results with a probabilistic interpretation. This is one of the benefits of these methods since often in real life problems there is not a single 100% correct answer to be found. Methods in this category include neural networks and other machine learning processes.

### State Of The Art

The computer Go programming community is relatively small compared to computer Chess but interest in the topic is building now that computer chess has reached such a high level of success. The central hub of this community can be found at the computer Go mailing list and the computer Go ladder within which programmers can enter their attempts in an ongoing tournament with most of the best programs available, including commercial programs [4,5]. The low number of entrants, around 20, perhaps reflects the difficulty of completing a Go playing program that can at least play competently at each stage in a game of Go.

### Many Faces Of Go

One of the best Go programs around is called Many Faces Of Go. This program uses traditional, hard AI techniques in combination with specialist knowledge databases and rule-based expert systems which are used with a deep game tree searching mechanism and a highly tuned evaluation function.

The weakness of this approach is that it is totally dependent on the accuracy and completeness of its internal database. Should a flaw be discovered by its opponent then manual reprogramming would be required to correct it.

### NeuroGo

One program that has been developed using neural network techniques is called NeuroGo [6].

The method used by NeuroGo when using the neural network is to reduce the board position into strings and empty intersections. The relationships between these units are used to construct a neural net.

The training target was determined by the Temporal Difference learning algorithm [15].

Several experts concerning relations between points and features on the board, relevant to determine the next move, are used in conjunction with the neural network. This is effectively feeding knowledge about important concepts in Go directly into the network, rather than attempting to make the neural net 'discover' these concepts itself.

### A NEW APPROACH

The foundation idea for this project was to use a neural network (NN) to suggest plausible moves and then pass these to a game tree analyser to search the moves using a traditional game tree search method. The neural network would thus act as a replacement for the expert systems used in Many Faces Of Go. The use of a NN for selecting possible moves should be fast and also allows the opportunity to expand the NNs knowledge in an automated fashion.

### Combination Of Hard And Soft AI Techniques

By attempting to use neural networks with game tree search this project is bringing together hard and soft AI. The main purpose behind this is to take the best features of each component and combine them to produce something that performs better than either of the two components separately. In the case of game tree search the advantage of using it comes from its ability to look ahead into the probable results of playing a particular move. The disadvantage is that it can be very resource intensive and inefficient.

The advantage of using neural networks is that they can provide some knowledge, learnt from experience, of actually playing the game. The disadvantage is exactly the benefit that game tree search provides, in that neural networks work with the position of the board as it is. They aren't directly able to handle looking ahead and considering the consequences of playing a particular move explicitly.

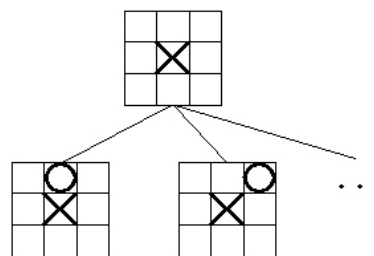


Figure 1: A Partial Game Tree for Tic-Tac-Toe

To summarise, the use of hard and soft AI techniques combined in a carefully thought out manner can negate the disadvantages of both and accent the advantages to produce an overall benefit to the system that it would not have been able to get through the use of only one or other of the classes of methods.

### Application To Standard Game Tree Search

A game tree search works by taking a root node, usually the current board position and then generating all possible lines of play from the root until a desired depth is reached or until some other cutoff criterion comes into play. Sometimes all possible moves are used, but it is common to cut this down by removing illegal and obviously bad moves from the tree. The terminal nodes in a game tree are assigned a score based upon some static analysis of the board position at that node. The scores from all the terminal nodes are filtered up the tree toward the root node, using a technique known as Minimax [12]. This way the best move, the one that maximises the player's score, regardless of the opponent's play, can be found. However the terminal node board scoring method must be fairly accurate else the minimax values will be irrelevant, and for many games the computational power required to generate a game tree large enough to produce a conclusive result is enormous. For instance the search space

size of Go has been estimated at around  $10^{170}$ , compared to Chess, which is about  $10^{40}$  [1].

In a standard game tree search much processor time will be spent on generating nodes for the next level of the tree but a neural network can be used to quickly and efficiently suggest the most plausible moves given a board position and this can be used at every level of the game tree, assuming the neural network is trusted enough to suggest reliable and high enough quality moves. Ideally around 6 moves would be used from the suggestions of a neural net interfaced to a minimax game tree algorithm, allowing a depth of about 7 or 8 ply to be reached with out too much stress on the system resources.

## SOFTWARE/HARDWARE DEVELOPMENT

The software system was developed with the following aims in mind:

- To provide facilities for the program to communicate with other Go playing programs. A generic neural network program should be developed to create and train neural nets.
- The generic application should be expanded and applied to Go.
- To process training data for the network an automated procedure should be developed.
- Utilities must be provided for board position evaluation, standard game tree search and methods for gathering Go specific information about the board position such as the number of liberties a string of stones may have.
- A graphical user interface should be provided for ease of use.

For communications the Go Modem Protocol was used [17]. An alternative to GMP which is currently under debate is called the Go Text Protocol, GTP [9].

The game tree search algorithm implemented is specifically a version of the classic Minimax search with alpha-beta pruning, called MTD(f) [11]. This is currently one of the best performing and most reliable Minimax type search algorithms developed. It was intended that this search algorithm should use the neural network module to supply moves for expanding the tree. Iterative deepening was also used, as this appears to be the most efficient method of expanding search trees. Several enhancements to the Minimax algorithm were also used including a transposition table, best move first and Enhanced Transposition Cutoffs, ETC [14]. Best move first is straightforward and means that, through the use of the transposition table, the best move at a branch in the search tree, from a previous search, will always be checked before other candidate child nodes. ETC is also simple, involving a simple check when a node is generated to see if the same node had previously caused a cutoff in another search and if so to check the cutoff condition again. Both of these additions allow for a considerable performance

boost to the standard Minimax algorithm, allowing much larger and faster searches to be performed.

A simple evaluation function for the Minimax search was used, just a liberty count and comparison for both sides.

## Design Issues

After some experimentation with different sized neural networks I found that a 3-layer network with 81 input neurons, representing 9x9 sections of a Go board centred on the proposed move, appeared to be the most useful. Only 1 output neuron was required to give a plausibility score for the move fed into the network and an arbitrary number of hidden neurons (in this case 45 neurons) were included. The highest scoring moves were selected to be used in the game tree search.

The back propagation algorithm for training was decided upon, mostly because of its general applicability to a wide range of problems [3].

Training data was acquired as a collection of professional tournament games in SGF format from the Internet [16].

An idea that occurred later in the project was of varying the granularity of consideration of the networks. Just as a human player might, the idea was to look at the board and narrow down an area in stages and eventually identify the best move to play. Until now the program had been designed to look at as many points on the board, and their neighbours in relation, as time and resources allowed. This new method would allow a succession of networks, trained to varying levels of detail to be used. And will be given some consideration later in this paper.

## IMPLEMENTATION

A simple GUI was developed which allows the user to control the neural network training and development. A Go board is present and games can be played against the program with it. The GUI is also intended to be used so the user can watch a game in progress between the program and another program via GMP or another protocol.

A useful test feature developed in the program was the game walkthrough test, which allowed the user to play through a professional game of Go that has been stored in Smart Game Format, SGF, the format the training data was initially found in [10]. The program itself makes suggestions and comparisons are made between the program and the actual moves made. Lots of information is generated at each move and outputted in a log window. Graphically, the board is shown and potential moves are coloured according to the plausibility score assigned by the neural network. This allows the user to get a general idea of the level of skill the network has learnt to.

Code to create and maintain GoString information was included, which monitors which stones belong to each string, what colour the string is and how many liberties the string has. This greatly increases the speed when detecting and removing captured groups.

The distilled experience of 36 professional tournament games was used in constructing a training database, producing over 50,000 training pairs, which pushed resource constraints to the limit. Due to the training set actually being fairly small in experience content, the networks were in danger of over training and becoming too specialised on the set of games used to produce the training set. Several issues concerning resources still have to be addressed before these problems can be overcome.

A coarser grained Area Finder network was trained for a reasonable amount of time and started to show some signs of sensible suggestions concerning the area of board to play in.

Two restricted move range networks were trained, both showing promising results. The first covered the first five ply moves and second covers from five ply to ten ply. An advantage of using restricted ranges is that the training is much quicker and the networks gain a higher degree of skill in their area than a more generalised network ever would.

The MTD (f) variation of the alpha-beta minimax search algorithm was implemented along with the aforementioned enhancements.

The expand function which creates child nodes given a position uses the 9x9 neural network to suggest a specified number of the most plausible moves as child nodes.

An alternative evaluation function for the Minimax search that was not implemented but could be a future path of research involves using a TD ( $\lambda$ ) trained neural network as an evaluation function. In fact this approach appears to be the most promising method of implementing an effective evaluation function, judging from research done by other parties [13].

## RESULTS & DISCUSSIONS

Looking at how long it takes various configurations to reach a move decision shows us some important points and in combination with some quality of result analysis can tell us to what degree the combination of soft and hard AI has been successful and if it is worth pursuing in the future.

First of all, looking at table 1, the difference between configuration 1, which used just a 9x9 network and configuration 2 which used a 9x9 network with an Area Finder network is easily explainable.

Configuration	Average Time Taken Per Move (seconds)
1. 9x9 Neural Network	0.784
2. 9x9 Neural Network + Area Finder	0.136
3. Alpha-Beta (Liberty Count)	40.712
4. 9x9 Neural Network + Alpha-Beta (Liberty Count)	9.02
5. 9x9 Neural Network + Alpha-Beta (Liberty Count) + Area Finder	1.56

TABLE 1 – SYSTEM CONFIGURATION SPEEDS

Using the coarser grained Area Finder network divides the board into 9 sectors and given the full 19x19 board selects the most appropriate sector out of the 9. Then the 9x9 network looks at all legal moves within that sector. For the first configuration all legal moves in the entire 19x19 board must be considered so we see a logical time difference of around a factor of 9. A similar affect is seen when comparing configurations 4 and 5. The use of the Area Finder network gives a substantial speed boost without adding any unreasonable overheads. If the quality of suggestions presented by the Area Finder network can be measured and built upon then this could be an effective and efficient method of incorporating neural network technology within a Go playing program.

The use of alpha-beta search added considerable computational overheads, however that is expected considering the nature of the algorithm. The liberty count evaluation function was used in all cases.

To assess and compare the quality of the neural networks and different configurations involving either or both soft and hard AI two approaches were taken.

Neural Network	Average Time Per Move	Percentage of time that actual move is in top n percent				
		10%	20%	30%	40%	50%
5x5	0.576	19.2%	36%	50.4%	59.2%	64.8%
7x7	0.736	16.8%	29.6%	41.6%	60%	65.6%
9x9	0.936	12%	30.4%	39.2%	52.8%	64%
Copy of 9x9	0.912	18.4%	34.4%	47.2%	56%	68.8%
11x11	1.256	9.6%	19.2%	30.4%	35.2%	44%
13x13	1.664	18.4%	28%	36%	51.2%	61.6%
Random 9x9	0.824	4.8%	20.8%	28.8%	35.2%	44.8%

TABLE 2 – NETWORK PERFORMANCE STATISTICS

The first method was used to determine the extent and level of training achieved by the neural networks. Several measures were used and information gathered about 6 different networks is displayed in table 2. The statistics were collected as each network played through the same professional game; the average time to select a move was recorded, as was the average rank of the actual move within all the moves considered by the neural networks. To give a more detailed look at the quality of the moves being selected the percentage of time that the actual move was ranked in

certain percentiles was also noted, for example for the 5x5 network the actual move was tanked in the top 10% of moves 19.2% of the time and was tanked in the 20% of moves 36% of the time. For comparison sakes a newly created, hence untrained, network was tested also, so we should expect, if training has worked at all, that the new network should have the lowest scores.

Looking at the results the first thing of note is that the larger the network the longer it takes to suggest a move. This is quite expected and reminds us that even though neural networks are fast compared to other AI techniques they can still easily build up a substantial overhead that must be kept in mind and minimised wherever possible. The best performing network appears to be the 5x5 with the actual move being ranked in the top 30% of moves just over half of the time.

Comparing all of these figures to the random 9x9 network shows that they all improved after training and reveals an interesting and very important point about over training. The ‘copy of 9x9’ network was an earlier version of the current 9x9 and has much better figures. This does suggest rather strongly that the current 9x9 has been over trained and the quality of its output has been degraded as a result. This also implies that a peak of training can be reached and by considering the figures they also suggest that the peak does not mean getting the very best move at rank number one. Rather it suggests, perhaps viewing it optimistically, that the network realises there may not be one perfect move but maybe lots of good moves and using a neural network allows the moves to be ranked effectively as opposed to selecting a single best move. This may be the strength of using neural networks in this instance.

A rather large hindrance that should be kept in mind is the time it takes to train a network. The 9x9 took around 2 months to reach a point where it started to over train. Larger networks and bigger training databases will take proportionately more time. This meant there was not a lot of spare time for experiments and trying alternatives, so I think this will be a stumbling block for quite a while in the future.

With the second quality measure the configurations used for timing an average move were used to play proper games of Go against GNUGo 26b [7]. The results are presented in table 3. It is important to play actual games since this was the original intention of creating such a program and is really the best way of judging its success in its intended environment. The program itself still unfortunately has a few problems and bugs that were given special provision. The program did not have any method to decide when to pass, so a game would continue until GNUGo passed or until a crash occurred.

Configuration	Game Score (we play black), (J= Japanese, C= Chinese)
1. 9x9 Neural Network	J: B-8, W-49 C: B-106, W-146
2. 9x9 Neural Network + Area Finder	J: B-3, W-21 C: B-54, W-71
3. Alpha-Beta (Liberty Count)	J: B-9, W-12 C: B-35, W-38 * Not Compete
4. 9x9 Neural Network + Alpha-Beta (Liberty Count)	J: B-9, W-25 C: B-107, W-124
5. 9x9 Neural Network + Alpha-Beta (Liberty Count) +Area Finder	J: B-7, W-18 C: B-55, W-66

TABLE 3 – GAME SCORES

Where a crash occurred it is marked in the results table as N.C. (not completed). When a game has reached an end, either on purpose or by fault, the board was scored by Jago, which functioned as arbiter between the programs [8]. A final point of note is that the program had no knowledge of the Ko rule and as such could have broken it and forfeited the game, however such a situation did not occur in any of the test games played.

If we look at the results of the test games against GNUGo we can observe several things from the scores. Both Chinese and Japanese scores are presented, with our program playing black for each game.

The first thing to note is that where the alpha-beta algorithm was included the score gap has been considerably narrowed. This would imply an overall improvement in defence and offence by the program thanks to the lookahead facilities provided by the alpha-beta routine. It also shows that although only a simple and occasionally probably inhibitory evaluation function was used, the liberty counter, a general improvement in play was fairly easy to achieve. Currently the alpha-beta is limited to a 6 move look ahead but it would probably be beneficial to make this a dynamic factor based on the line of play and resource availability.

Unfortunately the plain alpha-beta configuration would not complete an entire game so the scoring is pretty inaccurate and may not reflect the final state of the game had it reached its conclusion. This means we can see that the soft AI, neural networks, benefits from using hard AI, alpha-beta, techniques but we cannot be certain vice versa. It is possible that the neural networks may actually hinder the optimum play of the alpha-beta routine although this seems unlikely.

What more we can tell is that the addition of the Area Finder not only gives a speed advantage as discussed earlier but also increases the quality of suggestions. From observation of the games I would suggest that this is partly because a wider area of the board was played across when using the Area Finder than without, so the opponent found it harder to establish solid territories. When the Area Finder was not used the play tended to a single area and usually

stayed there throughout the game, allowing the opponent to establish unchallenged territories.

All these results show that soft AI has something to offer the problem area of Go, the limitations and extent of which require further investigation, however we have made some connections, various ideas have been tried and tested and a system that can support further research has been developed and implemented. More than anything else I think, questions have been raised and pointers for promising future investigations have been found.

There could be any number of ways to combine soft and hard AI. The trick is to do it in such a way as to maximise the strengths of each and minimise the weaknesses. If in doing so the combination is greater than the parts then the job is a success. From the results it seems clear that hard AI benefits soft AI and it is a pity the reverse cannot be concretely induced from the results collected but it would be reasonable to assume so.

## CONCLUSIONS AND FUTURE WORK

Many ideas cropped up throughout this project and given time and appropriate attention there are lots of avenues of research that warrant further investigation. Also the game of Go is such an expansive problem that there are many places where the system could be improved and alternative approaches could be taken. Amongst the best ideas that were considered are using various grain networks, how to combine them and which to use, limited range networks which tend to specialisation and of course other methods of combining soft and hard AI. There are many other soft AI techniques worth looking at, such as genetic algorithms and evolutionary programming and seeing how they could be combined efficiently and effectively with the more traditional hard AI methods. Also there are lots of hard AI techniques not considered here such as rule based systems that could benefit the soft methods, to give direction and guidance when faced with such a vast and chaotic wealth of information, which may be essential to give encourage the system to develop global strategies.

Interesting extensions to the ideas could include a closer look at the actual choice of neural network architecture and construction, and a more thorough review of the choices available. A deeper understanding of the way humans perceive and play Go could be developed from further work, perhaps leading to a better understanding of human pattern recognition. Go is an excellent environment that provides many opportunities for developing knowledge, not just of artificial intelligence and how to use it, but also of human intelligence and how the two compare, and it may also provide some impetus for the acceleration of development of soft AI techniques such as neural networks and other machine learning processes.

## REFERENCES

- [1] Allis, L.V., Van Der Herik, H.J., Herschberg, I.S., "Which Games Will Survive?", Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad, pages 232 - 243, Ellis Horwood, 1991.
- [2] Burmeister, J, "An Introduction to the Computer Go Field", and Burmeister, J., Wiles, J., 1995, "Associated Internet Resources", Available on the Internet at <http://www2.psy.uq.edu.au/~jay/go/CS-TR339.html>
- [3] Callan, R., "The Essence Of Neural Networks", Prentice Hall, 1999.
- [4] Ladder 2001, "Computer Go Computer Go", See <http://www.cgl.ucsf.edu/go/ladder.html>
- [5] Computer Go Mailing List, 2001, See <http://www.cs.uoregon.edu/~richard/computer-go/index.html>.
- [6] Enzenberger, M, "The Integration of a Priori of Knowledge into a Go Playing Neural Network", 1996, Available on the Internet at <http://www.uni-muenchen.de>.
- [7] GNU Go, latest version can be found at <http://freedom.sarang.net/software/gnugo/beta.html>
- [8] Grothmann, R, "Jago", 2001, available on the internet at <http://mathsrv.ku-eichstaett.de/MGF/homes/grothmann>
- [9] "GTP2001", Go Text Protocol information can be found 2/3 down the page at <http://freedom.sarang.net/software/gnugo/beta.html>.
- [10] Hollosi, A, "SGF User Guide", 1999, available on the Internet at [http://www.red-bean.com/sgf/user\\_guide/index.html](http://www.red-bean.com/sgf/user_guide/index.html).
- [11] Plaat, A, "MTD(f), A Minimax Algorithm Faster than NegaScout", 1997, available on the Internet at, <http://www.cs.vu.nl/~aske/mtdf.html>.
- [12] Owsnicki-Klewe, B, "Search Algorithms", 1999, available on the Internet at <http://www.informatik.fh-hamburg.de/~owsnicki/search.html>
- [13] Tesauro, G, "TD-Gammon, a self-teaching backgammon program, achieves master level play", Neural Computation, Vol. 6, No.2, 1994.
- [14] Schaeffer, J and A. Plaat, "New Advances In Alpha-Beta Searching".
- [15] Schraudolph, N, Dayan, P, Sejnowski, T, "Temporal Difference Learning of Position Evaluation in the Game of Go", Neural Information Processing Systems 6, Morgan Kaufmann, 1994, available on the Internet at <ftp://bsdserver.ucsf.edu/Go/comp/td-go.ps.Z>
- [16] Van Der Steen, J, "Go Game Gallery", 2001, available on the Internet at <http://www.cwi.nl/~jansteen/go/index.html>
- [17] Wilcox, B, "The Standard Go Modem Protocol - Revision 1.0", available on the Internet at <http://www.britgo.org/>