The Integration of A Priori Knowledge into a Go Playing Neural Network

Markus Enzenberger
markus.enzenberger@physik.uni-muenchen.de

September 1996

Abstract

The best current computer Go programs are hand crafted expert systems. They are using conventional AI technics such as pattern matching, rule based systems and goal oriented selective search. Due to the increasing complexity of managing this kind of knowledge representation by hand, the playing strength of these programs is still far from human master level. This article describes methods for integrating expert Go knowledge into a learning artificial neural network. These methods are implemented in the program *NeuroGo*. The network learns by playing against itself using temporal difference learning and backpropagation. The expert knowledge that is implemented at present in NeuroGo is simple compared with a conventional computer Go program. Despite of this, NeuroGo is able to achieve a playing strength which is equal to a conventional program playing at a medium level.

1 Introduction

The oriental game of Go is probably the most complex of all board games. However, its rules are simple. There are two players. Both alternately place stones of their colour (Black or White) on a board with 19×19 intersections (smaller sizes are possible). Black begins. The goal is to surround more territory than the opponent. Adjacent stones of the same colour form *strings*. A string can be captured if all its adjacent empty intersections, its *liberties*, are occupied by the opponent. An important pattern in Go is an *eye*. This is an empty intersection surrounded by a single string. The opponent can only play there if he captures the string and gains liberties by removing it. A string with two eyes is safe from capture, it is said to be unconditionally *alive*.

Despite of the success of computers in chess, the best current Go programs are still weak. Because of the large number of allowed moves, brute force search is not feasible. In addition, Go positions are difficult to evaluate. The security of strings from capture is often not clear, but can decide the outcome of the game. Conventional Go programs rely mainly on positional analysis. They use databases which contain thousands of patterns and rules. Some tactical subproblems are solved by search. This approach becomes more difficult to handle, the bigger the programs become. Side effects of newly integrated knowledge and unforeseen interaction of rules are a problem.

Therefore a hybrid approach which combines heuristic knowledge with machine learning is appealing. Schraudolph, Dayan and Sejnowski [5] have shown that a neural network can learn to beat a commercial Go program at a low level if special care is given to reflect the translational and colour symmetries of the pattern recognition task in the architecture of the network. However, they did not integrate any expert knowledge into the system which is inherent in conventional Go programs.

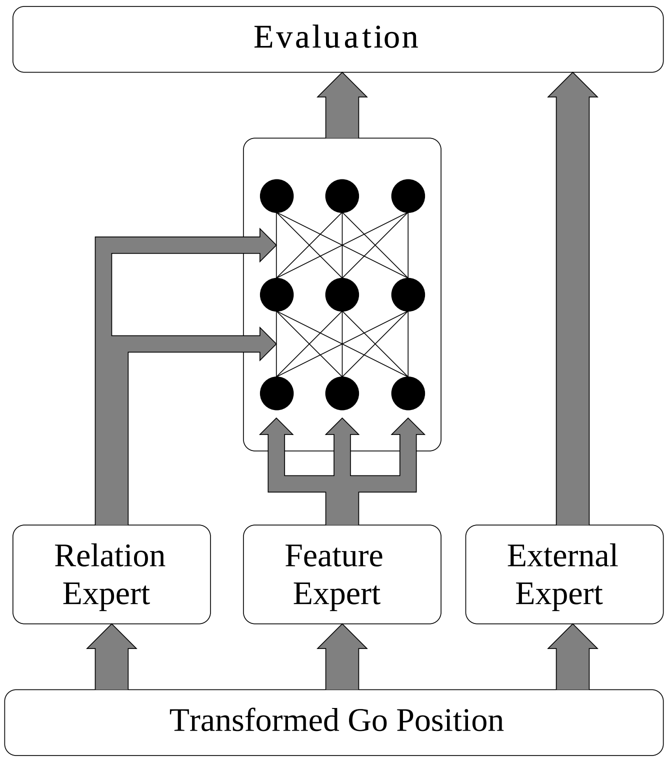

Figure 1: System architecture. The position is transformed into a set of strings and empty intersections. The feature expert calculates the network's input. The relation expert determines its connectivity. The external expert adds more knowledge, which is allowed to override the output of the network.

2 System Architecture

In the following the system architecture will be described. From an outside point of view, the system is given a board position and produces an evaluation for every intersection on the board. The evaluation is interpreted as the probability that this intersection will belong to Black at the end of the game. The architecture of the system is shown in figure 1. The board position is transformed in two ways. The system contains a neural network with one hidden layer and sigmoid activation functions. A priori knowledge is integrated by three different expert modules.

Transformations of the Position

The first transformation is that Black is always to move. If it is White's turn, the colours of all stones are exchanged. The second transformation is motivated by the fact that there is a strong correlation between the stones which belong to the same string. If the string is safe, all its stones will belong to the player at the end of the game. If it can be captured, then in most cases all the intersections that are occupied by its stones will become territory of the opponent. Therefore the given board position is not divided into intersections, but transformed into a set of strings and empty intersections. From now on, strings and empty intersection are called *units*.

Network

Because of the second transformation, the number of neurons in the input, hidden and output layer of the network depends on the board position. A static connectivity of the neurons is no longer possible. This is no handicap, the decision how to connect the neurons can be used for integrating more a priori knowledge, as will be described in the following section. In the network, there is one neuron per unit in the output layer and an arbitrary number of neurons per unit in the input and in the hidden layer.

Relation Expert

The relation expert uses a priori knowledge for detecting relations between a pair of units. A graph is constructed using the units as nodes and the relations as edges. This graph determines the connectivity of the network. There cannot be more than one relation per direction between two units. If the expert detects more than one, it must decide which the most important is. Two neurons of adjacent layers that correspond to two units are connected only if the units are related. The weights which are used for the connection depend on the type of the relation (including the type of the units) and the index of the neurons in the layers. They do not depend on the location of the units on the Go board.

This does not only incorporate translational and rotational invariances of pattern recognition on the Go board, but also invariances that are specific to Go programming. Go knowledge like "A string with two eyes cannot be captured" can be easily learned using this architecture, regardless of the form and size of the string and the location of the eyes. If care is given to identify only important relations, the relation expert can produce a network which is connected sparsely and efficiently.

Example

An example of the construction of the network by using the transformed Go position and the relation expert is shown in figure 2. A position on a 3×3 board which contains a single white string and two empty intersections is given.

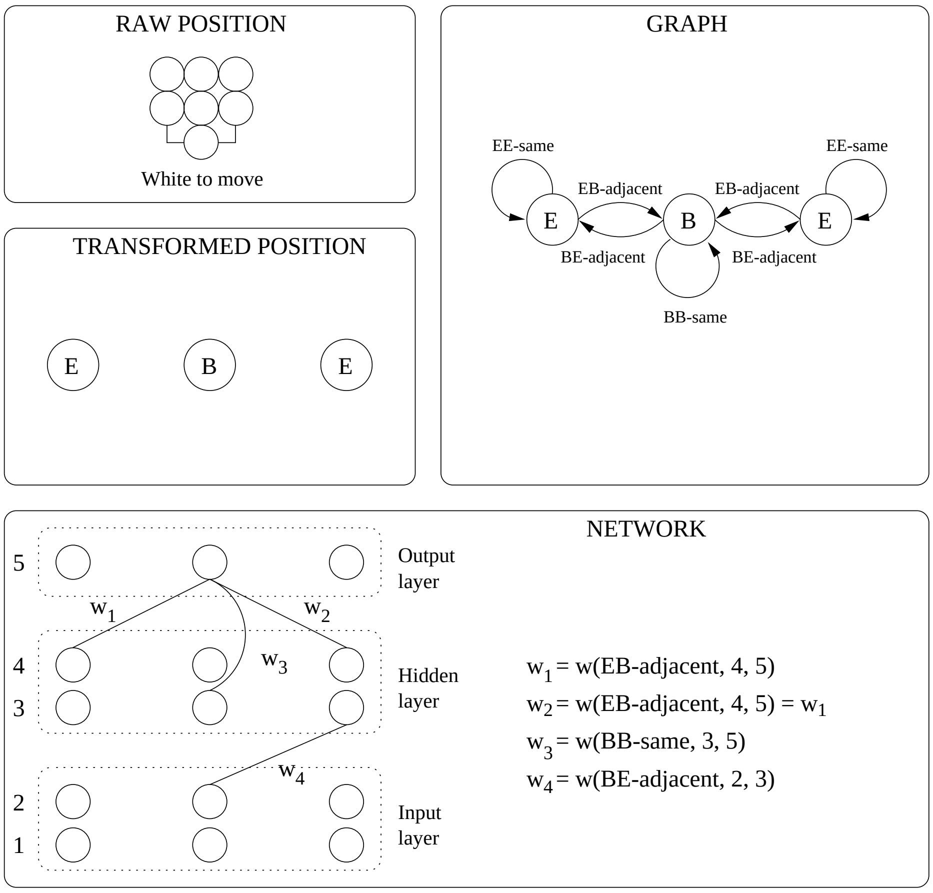

Figure 2: Example of the construction of the network. The relation expert recognizes the relations "same" and "adjacent". In this figure B stands for black string and E for empty intersection. Only a subset of all connections between the neurons is shown.

Feature Expert

The second method of integrating a priori knowledge is obvious and frequently applied in neural computation. Rather than feeding the network with the position only, the input is preprocessed by the feature expert. It detects features of a single unit and calculates the activation of the input neurons which correspond to this unit.

External Expert

The external expert recognizes and evaluates parts of the board completely by its own. It can be used if there is a simple and correct algorithm for evaluating a certain class of local positions. There is no need for a learning system then. The external expert overrides the output of the network. However, since the output of the system (together with the terminal position) determines the teaching signal in temporal difference learning (see below), the network will learn an evaluation function with a smooth interface to the evaluation of the external expert.

Playing and learning

Moves are chosen by playing all allowed moves of a position. The resulting positions are evaluated by the system. The move with the highest value is picked then. After the game has ended, the position is scored and played backwards. The training target is determined by the temporal difference algorithm TD(0) [6]: it is given by the output of the system for the position one move later or the terminal position if no moves follow. The weight changes are calculated by the backpropagation rule. Since a weight may occur many times in the network all weight changes belonging to the same weight are added. The weight changes are applied instantaneously after each target is presented to the network. All errors in the evaluation of a string unit are multiplied with its number of stones. This is necessary for learning to predict the overall score accurately.

3 NeuroGo

NeuroGo is a program which is based on the described architecture. It is participating in the *Computer Go Ladder* [4]. Because of the limited available computational resources and the need for patterns which are frequently occuring, the implemented knowledge is rather simple compared with conventional Go programs.

At present the following features are computed by the feature expert:

black or white string	empty intersection
number of liberties (1, 2, 3, 4, $\geq$5)	liberties of Black if he plays here (1, 2, 3, 4, $\geq$5)
number of stones (1 or 2, 3, $\geq$4)	liberties of White if he plays here (1, 2, 3, 4, $\geq$5)
can be captured in a ladder if string colour plays first	Black can be captured in a ladder, if he plays here
can be captured in a ladder if string colour plays second	White can be captured in a ladder, if he plays here
	eye for Black, n moves missing ($n = 0, 1, 2, \geq 3$)
	eye for White, n moves missing ($n = 0, 1, 2, \geq 3$)

A *ladder* is a deep tactical sequence during which a string cannot gain more than two liberties and is eventually captured or saved.

The relations that are currently implemented in NeuroGo reflect mainly the distance of two units, according to the fact that the correlation of two units decreases with their distance on the board. Two units have the distance Dn if the length of the shortest path between them that contains only empty intersections is n. In addition *groups* are detected. A group is a set of strings of the same colour which probably cannot be cut apart by the opponent. They are likely to form a single string in the game later on and are therefore strongly correlated. Two strings belong to the same group if they share two liberties or one liberty where the opponent will be captured in a ladder if he plays there. The relations are listed in the following table. They are sorted by decreasing importance.

	from black string	from white string	from empty intersection
to black string	same	D0 or D1	D0 (liberty)
	same group	D2	D1
	D1	D0 to group	D2
	D2		D0 to group
	D1 to group		D1 to group
to white string	D0 or D1	same	D0 (liberty)
	D2	same group	D1
	D0 to group	D1	D2
		D2	D0 to group
		D1 to group	D1 to group
to empty intersection	D0	D0	same
	D1	D1	D0
	D2	D2	D1
			D2

At present there exists one external expert in NeuroGo. Basically it is D. Benson's algorithm [1] [3] which can detect a certain class of unconditionally alive strings and territory. The extensions of this algorithm by M. Müller [3] are added.

4 Results

In an experiment, the performance of NeuroGo against a conventional Go program on a 9×9 board was tested. The size of the hidden layer was varied between 3 and 24 neurons per unit. The test opponent was the commercially available program "The Many Faces of Go" [2] (Release 2, Revision 8.03). This program was also used by N. Schraudolph et al. [5]. They came up with a network being able to beat this program at a low level (2–3) by using it as a training partner. To avoid over specialization to a single opponent, NeuroGo was trained only by playing games against itself. Some noise was introduced

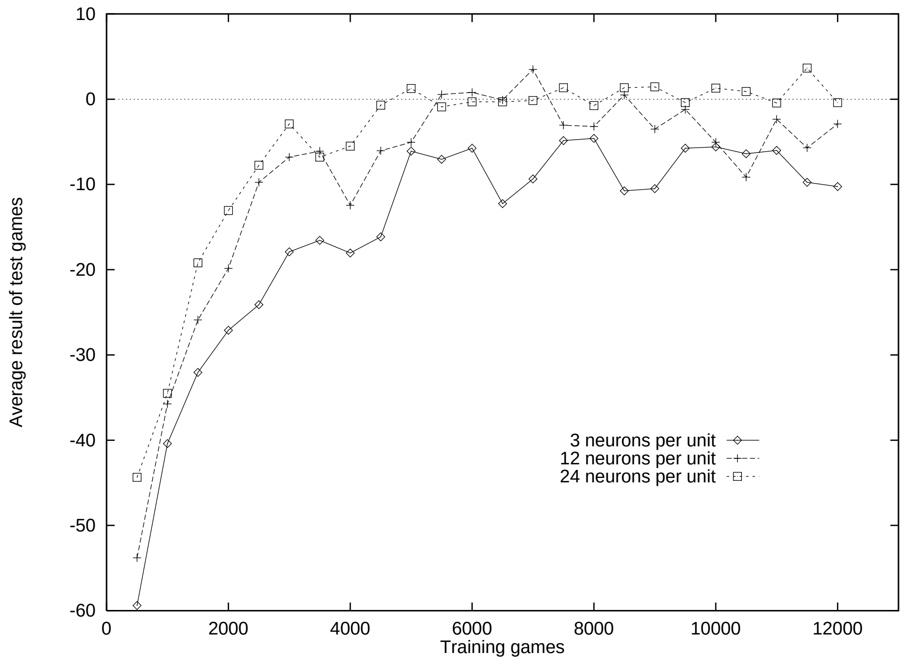

Figure 3: Training results. After 500 training games a set of 100 test games was played against the program "The Many Faces of Go" level 8. Networks of different sizes were trained.

by playing a move m with a probability P depending on the value v_m of the move. P is given by

$$P(v_m) = \frac{1}{N}\exp(v_m/0.35), \quad N = \sum_i \exp(v_i/0.35)$$

After a period of 500 training games, a set of 100 games against the opponent was played without training (50 with black and 50 with white). The medium playing level 8 (of 20) was chosen at the oppenent program.

As can be seen in figure 3 the largest network is able to achieve equal level of play after about 4500 games. It shows a more steady performance against the opponent than the smaller networks.

5 Conclusion

The test results show that the system architecture is suited to the learning task. The integrated knowledge is far less sophisticated than that of the opponent program. Therefore the abilities of the described architecture are not yet exploited. However, the integration of more expert knowledge will significantly increase the time of development. Large computational resources are needed especially if experiments will be performed on the usual 19×19 board. Furthermore it will be necessary to look for a way to integrate the evaluation function into a (selective) search strategy.

References

[1] Benson, D. (1976). *Life in the Game of Go.* Reprinted in: Computer Games II, Levy, D. (Editor), Springer, New York, 1988.

[2] Fotland, D. (1993). *Knowledge representation in The Many Faces of Go.* Manuscript available by Internet
`ftp://bsdserver.ucsf.edu/Go/comp/mfg.Z`.

[3] Müller, M. (1995). *Computer Go as a Sum of Local Games: An Application of Combinatorial Game Theory.* Dissertation, ETH Zürich. Available by Internet
`ftp://ftp.inf.ethz.ch/pub/publications/dissertations/th11006.ps.gz`.

[4] Petterson, E. (1994). *The Computer Go Ladder.* Internet World Wide Web Page
`http://cgl.ucsf.edu/go/ladder.html`.

[5] Schraudolph, N., Dayan, P., Sejnowski, T., (1994). *Temporal Difference Learning of Position Evaluation in the Game of Go.* In: Neural Information Processing Systems 6, Morgan Kaufmann 1994. Available by Internet
`ftp://bsdserver.ucsf.edu/Go/comp/td-go.ps.Z`.

[6] Sutton, R. (1988). *Learning to predict by the methods of temporal differences.* Machine Learning, 3:9-44. Available by Internet
`ftp://ftp.cs.umass.edu/pub/anw/pub/sutton/sutton-88.ps.gz`.