Evolving Neural Networks to Play Go [*][†]

Norman Richards, David E. Moriarty, and Risto Miikkulainen
The University of Texas at Austin
Austin, Tx 78712
orb,moriarty,risto@cs.utexas.edu

Abstract

Go is a difficult game for computers to master, and the best go programs are still weaker than the average human player. Since the traditional game playing techniques have proven inadequate, new approaches to computer go need to be studied. This paper presents a new approach to learning to play go. The SANE (Symbiotic, Adaptive Neuro-Evolution) method was used to evolve networks capable of playing go on small boards with no pre-programmed go knowledge. On a 9×9 go board, networks that were able to defeat a simple computer opponent were evolved within a few hundred generations. Most significantly, the networks exhibited several aspects of general go playing, which suggests the approach could scale up well.

1 Introduction

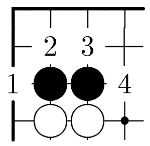
Go is hard. For computers at least, this is true. Though the game has not received the level of attention that computer chess, for example, has received, considerable effort has gone into trying to create strong go playing programs. Yet, despite this effort, the best computer programs are still relatively weak.

There are a number of reasons why go is hard for traditional computer game playing techniques: the branching factor is prohibitively large, the game is pattern oriented, and there are multiple interacting goals. In fact, the game is so difficult that new techniques are probably going to be needed before go programs are as strong as those that play checkers, chess, or Othello.

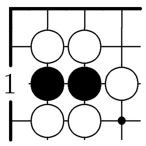
This paper explores the usefulness of neuro-evolution as a mechanism for learning to play go. The SANE (Symbiotic, Adaptive Neuro-Evolution [7, 8, 9]) algorithm demonstrates that networks that display a general ability in playing go on small boards can be evolved without

[*]To appear in *Applied Intelligence.*

[†]This research was supported in part by NSF under grant #IRI-9504317.



(a) Four liberties.



(b) One liberty.

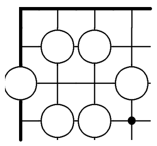

(c) No liberties.

Figure 1: The group in (a) has four liberties, or adjacent free positions, while the group in (b) has one. After that last liberty is lost (c), the group is said to be captured and is removed from the board.

any prior knowledge about the game. This result forms a promising foundation for scaling up to full-scale go.

The paper begins with an introduction to the rules of go followed by a brief word on computer go and why neural network techniques might be useful for go programs. Next the SANE neuro-evolution algorithm is is reviewed, and details of the architecture and the experiments given. The paper concludes with an analysis of the strategies evolved and suggestions for future research.

2 The Game of Go

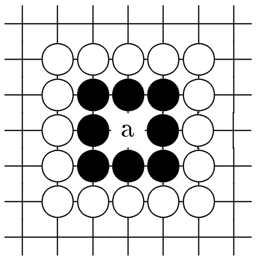
Although the term go is taken from the Japanese word for the game, go is believed to have originated in China more than 3,000 years ago, making it one of the oldest board games still actively played in modern times. Go is an appealing game because it appears simple yet features strategy and tactics that rival games such as chess.

Go is played on a square grid 19 intersections across. Smaller boards are often used for teaching purposes. The two players, black and white, alternate placing stones of their respective colors on the intersections of the grid. Game play continues until both players pass, at which time the score is calculated and a winner is determined.

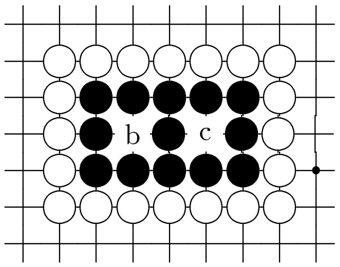
Game play is deceivingly simple. Stones can be placed on empty intersections. Once played, a stone cannot be moved to another location. However, a stone or a group of stones can be captured and removed from the board.

A liberty is an empty point adjacent to a group of stones. Any group that has no liberties is said to be dead and the stones in that group will be removed from the board. For example, the black group in figure 1a has 4 liberties. Figure 1b shows the same black group with further white stones placed such that the black group now has only 1 liberty. If white were to play an additional move at 1, the black stones would be reduced to 0 liberties. They would be considered dead and removed from the board, as shown in figure 1c.

The liberty rule gives rise to the simple concept of an eye. Any group of stones that completely surrounds some interior space is said to have an eye. In figure 2a, the black group has one eye, at point "a". The black group in figure 2b has 2 eyes, at points "b" and "c". The first group can easily be captured if white plays at point "a". However, the second black group cannot be captured by white as white would need to simultaneously occupy



(a) A group with one eye.



(b) A group with two eyes.

Figure 2: Eye space determines life and death of a group of stones. The group in (a) has one eye and can be killed by placing a stone in the eye (position "a"), while the group in (b) has two eyes, "b" and "c", and cannot be killed.

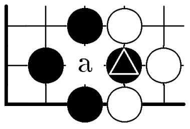


Figure 3: Repetition of full board positions is not allowed. If black has just played the marked stone to capture a white stone at "a", white would not be allowed to play at "a" immediately to capture the black stone, because that would recreate the board full board position before the black's move.

both points "b" and "c" to reduce the black group to 0 liberties. This is not possible, and therefore the group cannot be killed. This example demonstrates the most common and simplest form of a living group. Forming living groups is one of the primary goals of the game.

Previous full-board positions cannot be repeated. This rule is known as the ko rule. Figure 3 shows an example of how the rule applies. Suppose black has just played the marked stone to capture a white stone at point "a". White would not be allowed to immediately play at point "a" to capture the marked stone because it would recreate the board position before black's move. White must instead play elsewhere, and thereby create a new full-board position. If black does not move to point "a", white would then be allowed to play at "a" on a subsequent move because the full-board position would no longer be repeated.

Play continues until there are no more moves of value to be made and both players pass. Each side receives a score where all stones and all locations completely surrounded by groups of the same color are counted as points. A komi, 5.5 points in a typical even game, is added to white's score to offset black's first move advantage. The player with the highest score wins. Because komi is not an integer, a tie cannot occur.

Unlike many other board games, go provides an easy way to handicap games so that players of different ranks can play an even game. White will give black a certain number of free moves at the beginning of the game. This advantage is generally well defined, and go ranks are based on it. A player who is ranked 2 stones above another player should be able to give the other player 2 free moves in order to play an even game.

3 Computers and Go

Despite the relatively simple game play, computers have had little success mastering the game. Whereas in games like chess, Othello and checkers the traditional game playing techniques such as minimax search and its variations are competitive even at the master level [4, 5, 13], those techniques, when applied to go, have not been able to produce programs that can challenge even weak amateur players. The best computer programs in the world are ranked 6–8 stones below what would be considered master level. Progress continues to be made; however, the gap is so large that traditional techniques are unlikely to reach even the weakest master levels for some time to come.

A game of go can be divided into three general phases: the opening, the midgame and the endgame. Computers have had varying success in each of the these stages, revealing insight into what can and cannot be achieved with computational methods.

3.1 The Opening

The opening stage of the game is referred to as the fuseki. Typically play starts in the corners. Specific sequences in the corners are referred to as joseki, and they are similar to book openings in chess. However, the fuseki typically refers to the direction of play as it relates to all 4 corners. Good go programs tend to have joseki move databases that range from 5,000 to 50,000 moves, and current programs do not have any difficulty in playing through a database of joseki sequences. Choice of joseki and choosing between variations is more troublesome; however, play by the computers is not advanced enough to consider such issues. Even with such limited techniques, some programs are capable of playing very good openings.

3.2 The Midgame

As play moves into the midgame, search-based programs begin to have difficulty. One reason is the sheer size of the game. On a 19×19 board, there are typically 200-300 potential moves available from a midgame position, so brute force searching of the game tree is not a viable option.

Current go programs apply a wide variety of techniques to control the complexity of the midgame. Typically, a move generation facility is used to generate a number of candidate moves from a position using techniques such as pattern libraries, tactical analysis, and rule-based expert systems. Then, the candidate moves are evaluated, usually through static board evaluation functions. Some programs rely solely on the evaluation function for choice of moves while others attempt limited global search using traditional search techniques. Search plays a more prominent role in local tactical analysis where the number of moves and the size of the search tree are significantly smaller.

There are several problems with the current midgame techniques. They tend to be difficult to apply and error-prone. Because most evaluation techniques are static, it is difficult to achieve general play, and advance can only be made by laborous human tweaking of rules, patterns and databases.

3.3 The Endgame

The endgame is typically significantly easier for computers. During the endgame, programs are capable of playing very well because the branching factor is significantly reduced and patterns are local in nature. However, in the endgame the score is often already mostly decided and fewer points are usually at stake. Even perfect endgame play might only change the score by a small number of points. So, the most important and most difficult part of playing go is in the midgame where the traditional techniques are the weakest.

4 Neural Networks and Go

Go is largely pattern based; as a matter of fact, go players often refer to board positions as shape. Groups of stones can be said to have good shape or bad shape depending on the shape's potential of creating a living group and of efficiently capturing territory. Human players instinctively know where to search for moves based largely on learned knowledge of shape. Although there are many techniques that highly skilled players use and computer programs do not, viewing the board as a search node instead of a collection of shapes and patterns is probably the most significant factor holding computer go programs back.

Neural networks are very good at pattern transformation tasks, and thus could well be applicable to go. A network could be trained to compute a mapping between the input space, that is, the current board position, and the output indicating the next move. The main problem with this approach is the credit assignment problem. Suppose a standard backpropagation neural network [12] were being trained to play go. For backpropagation to work, advance knowledge about the best move at any given position would be required. Such knowledge is difficult to come by. In reality, only the final game result is available. The credit assignment problem is the problem of determining which of the many moves played were good and deserve credit for a win, and which were bad and deserve to be blamed for a loss. In go, this problem is severe enough that standard learning techniques such as backpropagation cannot be effectively applied.

5 SANE

SANE[1] (Symbiotic Adaptive Neuro-Evolution [7, 8, 9]) solves the credit assignment problem by using evolutionary algorithms to search for effective neural networks. Instead of punishing or rewarding individual moves, networks are evaluated, selected, and recombined based on their overall performance in the game. Evolutionary algorithms perform a global, parallel

[1]SANE is described in more detail in [8], and the source code can be obtained from `http://www.cs.utexas.edu/users/nn/`.

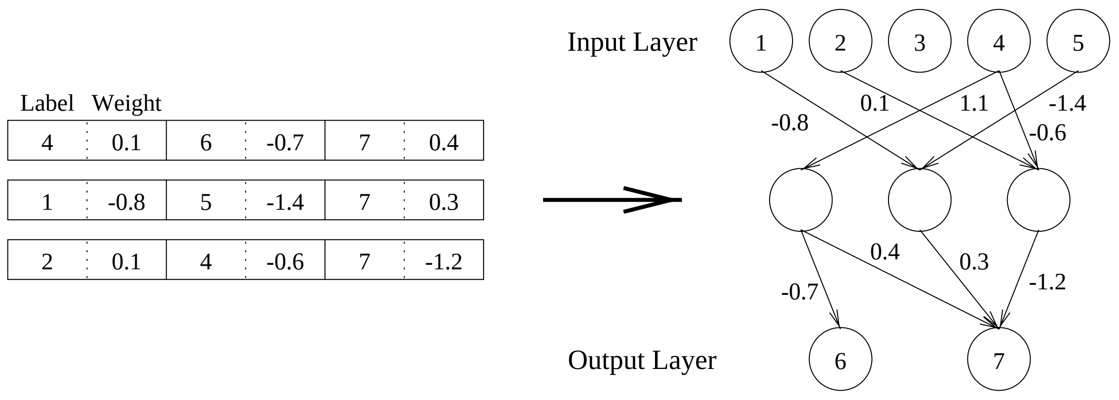


Figure 4: A three-layer feedforward network is created from 3 neurons. The neurons are shown on the left, and the corresponding network is shown on the right. Labels indicate which input or output unit a connection corresponds to, while the weight indicates the strength of the connection.

search and are guided by a fitness function that measures the goodness of a particular solution. The search tries to maximize the goodness level throughout the search space to find the best solution.

SANE differs from other approaches to neuro-evolution systems where each individual in the population represents a complete neural network. In SANE, two separate populations are maintained and evolved: a *population of neurons* and a *population of neural network blueprints*. The neuron level evolution explicitly decomposes the search space and maintains a high level of diversity throughout evolution. The blueprint population maintains and exploits effective combinations of individuals in the neuron population. Conjunctively, the two levels of evolution provide an efficient genetic search that is capable of solving difficult real-world decision problems with minimal domain information [8].

In the neuron population, SANE evolves a large population of hidden neuron definitions for a three-layer feedforward network (figure 4). A neuron is represented by a series of connection definitions that describe the weighted connections of the neuron from the input layer and to the output layer. Each neuron has a fixed number of connections, but may allocate them arbitrarily among the units in the input and output layers. A connection definition consists of a label and weight pair. The label is an integer value that specifies a specific input or output unit, and the weight is a floating point number that specifies the strength of the connection. Figure 4 gives three example hidden neuron definitions and the resulting neural network.

The activation of a neuron is computed as the sum of all the connected input units multiplied by their weights and passed through the sigmoid activation function $\sigma(x) = 1/(1 + e^{-x})$. For this application, the output units are linear so that both positive and negative values and be generated.

Neural networks could be formed by randomly choosing neurons from the neuron population. In fact, this approach performs well in simpler problems [7, 10]. However, random participation does not retain knowledge of the best combinations of neurons and can often

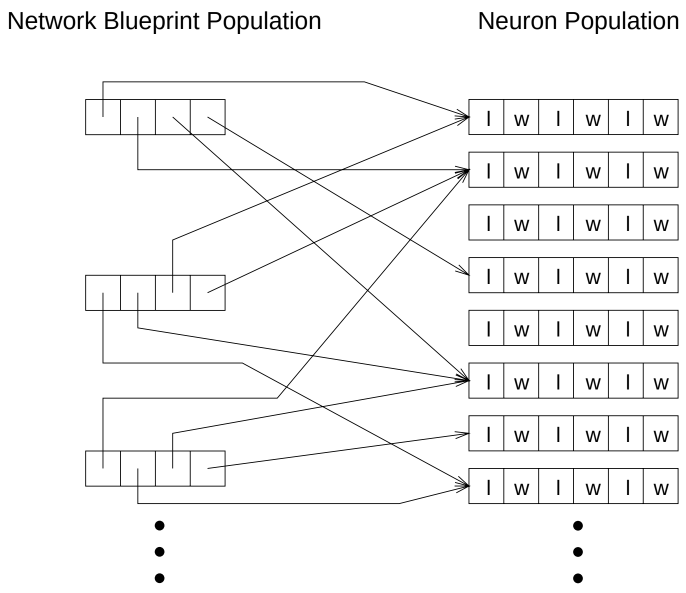


Figure 5: The network blueprints consist of a set of neurons in the neuron population. A neural network is formed from a blueprint by following its neuron pointers and decoding the respective neurons.

stall the search in more difficult problems [8]. To focus the search on the best neuron combinations, SANE maintains and evolves a separate population of good neuron combinations called neural network blueprints. The blueprints are made up of a series of pointers to members of the neuron population and define an effective neural network from a previous generation. Figure 5 shows how the network blueprint population and the neuron population are related.

SANE integrates the neuron and blueprint populations in a generational evolutionary algorithm that iterates over two phases: an evaluation phase and a reproduction phase. In the evaluation phase, SANE simultaneously evaluates the blueprints and the neurons. A blueprint is evaluated by the performance of the network that it specifies. A neuron is evaluated based on the performance of the networks in which it participates. The basic steps in the SANE evaluation phase are shown in the following pseudo code:

```
for each neuron $n$ in population $P_n$
    $n.fitness \leftarrow 0$
    $n.participation \leftarrow 0$
for each blueprint $b$ in population $P_b$
    $neuralnet \leftarrow \mathbf{decode}(b)$
    $b.fitness \leftarrow \mathbf{task}(neuralnet)$
    for each neuron $n$ in $b$
        $n.fitness \leftarrow n.fitness + b.fitness$
        $n.participation \leftarrow n.participation + 1$
for each neuron $n$ in population $P_n$
```

```
    $n.fitness \leftarrow n.fitness$ / $n.participation$
```

Neural networks are formed from each blueprint and evaluated in the task environment. The evaluation score is given to each blueprint and is added to each neuron's fitness variable. After all blueprints have been evaluated, each neuron's fitness is normalized by dividing the sum of its scores by the total number of networks in which it was a participant.

In the reproduction phase, SANE uses common genetic operators such as selection, crossover, and mutation to obtain new blueprints and neurons. Each population is ranked based on fitness and a mate is selected for each of the *elite* individuals. In this application, the elite parameter is defined as the top 15% in the blueprint population and the top 25% in the neuron population. The mate for each elite individual is selected from the other elite individuals. SANE uses a one-point crossover to mate two individuals, which creates two offspring. The offspring from each of the crossover operations replace the worst performing individuals (according to fitness) in the population. All individuals that are not explicitly replaced by offspring remain in the population, although they may be mutated.

A conservative mutation rate of 1% per chromosome position is used on the neuron population, because neuron evolution automatically maintains high diversity (good networks require serveral different types of neurons). A more aggressive, two-tiered strategy is used on the blueprint level. First, a small number (approximately 1%) of neuron pointers in each blueprint are swapped with randomly selected neurons in the neuron population. Second, pointers to breeding neurons are replaced by pointers to their offspring with a 50% probability. The second mutation component promotes utilization of offspring neurons, which has two advantages. First, it creates diversity in the blueprint population, and second, it explores new structures created by the neuron population.

6 Applying SANE to Go

SANE has previously been shown effective in several sequential decision tasks including robot control [7, 8, 9], constraint satisfaction [10], pursuit and evasion [3], and the game of Othello [6, 8, 10]. This paper will evaluate the usefulness of SANE in learning to play go. SANE is used to evolve networks to play on small boards against a simple computer opponent, and the scale-up properties are evaluated.

In order to apply SANE, three aspects of the architecture must be specified: the network parameters, evolution parameters and the evaluation function. Let us look at each one of these in turn in the go task.

6.1 Network Parameters

SANE evolves standard three-layer feed-forward networks. The network architecture is fixed; only the associated weights and connections are evolved. The number of units depends on the board size. There are 2 input units and one output unit for each board position. The

Board size	Neuron Population	Blueprint Population	Number of neurons per network
5×5	2000	200	100
7×7	3000	200	300
9×9	4000	200	500

Table 1: Network definitions used for evolving networks for various small board sizes.

first input unit indicates whether a black stone is present at that location, and the second unit whether a white stone is present. Since only one stone can occupy any given board position at a time, both input units cannot be active simultaneously for any position.

The output units are signed floating point values. A positive value indicates a good move. The larger the value, the better the move. Negative (or 0) output indicates that the move is not suggested.

6.2 Evolution Parameters

Most aspects of SANE are easily tunable. Some experimentation was done to find good values, however, it was not necessary to find optimum values as SANE operates well as long as the values are withing reasonable ranges.

The neurons evolved are 312 bits long and represent a set of 12 weights connecting either from input layer to hidden layer or from the hidden layer to the output layer. Table 1 shows the population and network sizes used in conjunction with the various board sizes. Each generation, 200 networks were formed. This allowed each neuron on average to participate in 10-25 networks per generation. Mutation occurred at a rate of 0.1% The crossover operation was a one-point crossover between neurons or networks in the breeding population. The top 25-30% of the population were allowed to breed.

6.3 Evaluation Function

The most difficult aspect of the evaluation function was deciding on a set of evaluation criteria that could be computed completely without human intervention. The first difficulty is in determining the end of the game. When humans play, the end of the game is decided by agreement. When the players feel the game is over, they pass their turn. Stones that are mutually agreed to be dead are removed from the board. If there is a dispute, play can be resumed to settle the issue. After the status of each group is determined, a final score is calculated.

Since there is no separate output unit for pass, the network can pass only when none of its positive output units (if any) correspond to a legal move. Because there is no arbitration phase for disputed groups, a series of 3 passes is required to end the game. This simplifies certain endgame situations where ko (i.e. repetition) might occur.

Removing dead stones is more difficult. Rather than defining a separate protocol for this task, the evaluation function requires a player to explicitly kill any stones it thinks are dead. Human players find this process tedious, so those groups that are obviously dead are simply removed from the board at the end of the game. For computer opponents, killing groups is not so tedious. If all stones on the board are considered alive, the need for settling disputes after the game is over is eliminated, and the task of scoring is greatly simplified.

An upper bound is placed on the number of moves, so that it is not actually necessary to check for repetition of entire board positions. It is enough that only the simple ko (demonstrated in figure 3) is checked. An upper bound also ensures that non-repeating but prohibitively long sequences are not followed. Games between human players do not involve such sequences. However, they may occur in a game by an unskilled program. An example of such a sequence would be the filling in of a player's own eye space, which allows a previously alive group to be killed. If two unskilled opponents play in this manner, excessively long sequences of play might result. Such play is punished by counting excessively long games as a loss for the network. Because this behavior is selected against, the networks become less likely to develop it.

Since all stones still on the board are presumed to be alive, determining the score becomes a straightforward task. Simple Chinese scoring, where all stones of each color and all locations completely surrounded by stones of that color are counted as points, is used.

The evaluation function must produce a fitness level for the network, and it should be a fine-grained value so that slight improvements in the network's play can be rewarded. In our experiments, the difference in score between SANE and its opponent (for example +10.5 or -7.5) is summed over N games, which allows for good resolution in determining improvement.

7 Results

SANE was tested with various board sizes. The opponent used was wally (written by Bill Newman), a simple publicly available go program. Wally is a good choice for an opponent for several reasons. First, wally is one of the few go playing programs available in source code. This turns out to be particularly helpful when trying to adjust parameters, like the degree of randomness, to make the opponent more useful as a training partner. Second, wally's skill level is appropriate for a first training partner. It is strong enough to be a challenge to an unskilled network without being so strong that progress cannot be made.

7.1 Evolution Efficiency

SANE was able to evolve a network that could defeat wally on small boards. On a 5×5 board, SANE needed only 20 generations, on a 7×7 board, 50 generations, and for a 9×9 game, 260 generations. These numbers are averages over 100 - 1,000 simulations, requiring the ability to beat the opponent 75% of the time. The network was playing black without a komi, which is an equivalent to a 1-stone handicap for the network.

Although these results were relatively easy to get, they take a lot of CPU time (up to

5 days for the 9×9 board). Moreover, the training times increase with board size quite rapidly. It can be estimated that for a 13×13 board, several thousand generations would be required, and for a full-size 19×19 board, perhaps tens of thousands. The CPU time for such simulations could be more than a year with the current CPU speeds, and was not available. However, it is still possible to get an idea of how well such a network plays go by studying the effect of nondeterminism and handicap in the opponent.

7.2 Effect of Nondeterminism

An important issue with developing general go playing is the degree of determinism of the opponent. SANE actually manages to learn to defeat more deterministic opponents very rapidly. However, in those cases the network learned little about playing go and only learned what was necessary to win against that particular opponent.

To force the network to learn more diverse solutions, 10% non-determinism were applied to wally. This means that 10% of the time, instead of making the normal move, a random legal move would be played instead. The 10% value was chosen experimentally to be a reasonable value. Smaller values did not significantly increase the diversity of the games played nor the solutions learned, and larger values made the opponent behave too randomly and easy to beat.

As a test of generality, one network was evolved against the original wally on a 7×7 board, while another was evolved against wally playing with 10% randomness. An otherwise deterministic player playing occasional random moves should be weaker in absolute terms. However, when playing a series of games against a learning opponent, the deterministic player turned out to be easier to beat. The first network learned to defeat wally very rapidly. However, it would be defeated easily by the weaker but less-deterministic wally. In fact, it would even lose some games against the randomly moving opponent. The network's behavior in this case was not diverse enough to be useful against other opponents. Instead of learning moves that represent general go-playing ability, the network simply learned tricks and simple sequences that utilize flaws in the static opponent.

The network playing against the less deterministic opponent required more generations to train. However, the solution evolved was capable of defeating wally at various levels of determinism, including its normal mode of play, and did not lose to the random opponent.

7.3 Effect of Handicaps

Since few stronger go playing programs are freely available, there was no good opportunity to evolve networks against other opponents. However, the go handicapping mechanism does provide a way to modify the difficulty of the game against a given opponent.

Networks were evolved on the 7×7 board while giving wally differing handicaps. Initially, the networks were evolved to play black and make the first move. After about 50 generations, a network evolves to defeat wally. The networks were then evolved with wally playing the first move. After 130 more generations, a network was able to beat wally 75% of the time.

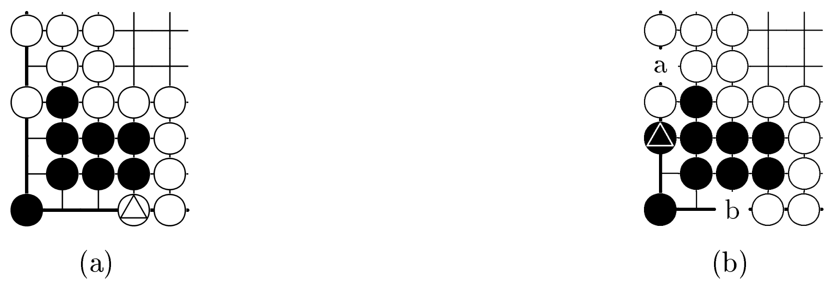


Figure 6: This position is from a game played between wally (white) and the network (black). After white plays the marked stone in (a), black should be dead in the left corner as it should not be able to make the two eyes necessary to live. But the network has learned a trick. It plays the marked stone in (b), threatening to capture the white stone below “a”. Instead of playing at “b”, which would ensure the death of the black group, it plays the move at “a” to defend the single stone below it. As a result, the network can move to “b” and live.

At this level, it could give wally a 2-stone handicap and still win 45% of the games.

With these results, we can get a rough idea of the level of play evolved. Handicap stones on a small board represent a larger difference in skill than on a larger board. For example, a single handicap stone on a 13×13 board represents approximately 3 stones on a 19×19 board. On a 9×9 board, the difference is about 4–5 stones. Thus, the 2 stone handicap on a 7×7 board may represent a difference in skill of about 10 stones, which is quite significant and would allow a good amateur compete with master-level player on the full board. Although this is just a rough estimate, it shows that quite powerful go play can be achieved through neuro-evolution methods.

8 Strategies Evolved

Given that the networks started with no prior knowledge on how to play go, an important question is: what kind of strategies did they evolve?

One peculiar problem with the evolutionary approach is that the strategy evolved often exploits weaknesses found in the particular opponent rather than representing good general go playing abilities. Figure 6 is an example of such a situation taken from an actual game played by a network against wally on a 7×7 board. The network is playing black and wally is white.

In this situation, white plays the marked stone in 6a. This is a move that should effectively kill the black group in the lower left corner because the black group would no longer be able to make two eyes. However, black has learned that it can actually win in this situation against wally. It plays the marked move in 6b, which makes a single eye and threatens to capture the single white stone above it at “a”. The correct move for white is to play “b” next. Allowing black to play at “b” would give black life. However, this particular computer opponent does not realize this and picks the tiny defensive move at “a” over the large offensive move at “b”. The network has learned to take advantage of this weakness and moves to “b” as its next move.

That such strategies would evolve is understandable considering that the opponent is the only source of information the network has about the game. The network is never explicitly taught about living or dead groups. It's concept of a living group is any group that the opponent cannot kill. In this case, since the opponent cannot kill this group, the network learns it as a favorable position. This example emphasizes the importance the training partner has on the strategies learned.

One possible method for forcing the network to rely less on exploiting this type of weakness in specific opponents is to evolve against a variety of opponents. The network would be less likely to learn these kinds of techniques because it is less likely that the same flaw will be present in multiple opponents. This type of evolution would have a better chance of producing a well-rounded go playing program. However, the lack of multiple freely available go playing programs makes this approach impractical at present time.

The neuro-evolution system is clearly learning enough to defeat a simple opponent, but are the networks evolving to play go in any general sense? A closer inspection of game transcripts shows that especially when evolved against a nondeterministic opponent, the networks demonstrated a reasonable amount of diversity and were able to cope with variations in play from the opponent.

At the beginning of evolution, the networks' outputs are essentially random. After a few generations, they start to make simple living groups. Typically, they evolve the capability to make one or two such groups along the edges or in the corners, and to extend them from there. As evolution continues, the networks become more flexible and capable of developing a greater variety of living positions. Such a strategy is valid, although not particularly strong. Since this type of strategy is all that is required to defeat the computer opponent, the network really does not need to develop more advanced strategies. Against more powerful opponents, the situation would be different. The experiments with handicaps show that in such cases, more powerful strategies are likely to develop.

Some well-known general go strategies were also evolved. For example, consider choosing the first move. In the first few generations, the network plays quite randomly, and therefore its first move tends to be on the edge or the outer lines of the board since they comprise 56 of the 81 positions on a 9×9 board. Such a move is not a good idea, however, because it is likely to lead to a losing position. Indeed, in a few generations the networks start to make more opening moves near the center of the board. Since the earlier strategy led to losses, the networks that did not use that strategy are now more prevalent in the population. Later on in evolution all the best networks open at or near the center of the board, which is exactly the strategy good go players use. Remarkably, the evolution system discovered it entirely on its own, based on what moves led to wins and losses. This result suggests that the neuro-evolution method is capable of developing good go playing strategies without preprogrammed knowledge, directed by the sparse reinforcement of the game outcomes only.

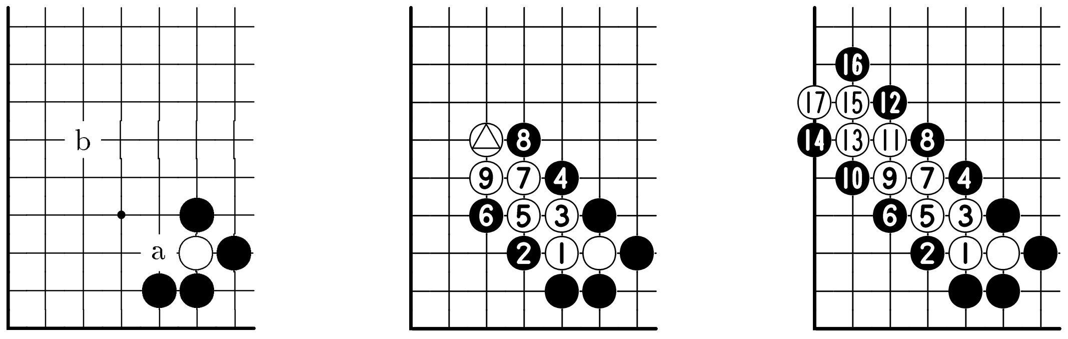


Figure 7: Figure (a) shows a position known as a ladder, which retains its shape as it is grown in successive moves. The life or death of the white stone depends on groups far away on the board. Figures (b) and (c) show two such situations: in (b) the white stone in location "b" allows white to escape. In (c) the ladder reaches the edge of the board and the white group is killed. In real games, such variations can span the whole board and are difficult to evaluate with only local methods.

9 Future Work

There are three main issues for future work: scaling up to larger boards, enhancing the network architecture, and evolving against stronger opponents.

9.1 Larger Boards

Ideally, a network should be able to play on any board size. Currently the networks can only play on the board on which they were evolved. For example, a network that was evolved on a 9×9 board is not able to play on a 7×7 board. One possibility would be to design a representation that is independent of board size. Another would be to evolve solutions that only consider a portion of the board at a time. This type of evaluation function could then be extended to cover boards of varying sizes.

However, considering only local board positions instead of the whole board tends to result in weaker play. To see why, consider the position shown in figure 7, known as a ladder. Based on only the local position, it is impossible to tell whether or not the white group can escape. White can play at point "a", and may live or die depending on stones that are on the other side of the board. If there is a white stone at point "b", for example, white can easily live. The extra stone allows white to break out of the ladder, as can be seen in figure 7b. However, if there are no stones on the area, white cannot live. Eventually the ladder position faces the edge of the board, where it is a losing position for white as can be seen in figure 7c. This way stones that are far away from the current play can transform the position drastically. These types of positions can span the entire board, not merely one corner. Recognizing such distance relationships is essential for playing go on larger boards, yet they cannot be captured with methods that consider only part of the board at a time.

It is likely that other types of network architectures need to be employed before play on full boards becomes practical. Possibilities include architectures that use preprogrammed features or are hierarchically organized,

9.2 Advanced Architectures

Evolving simple unstructured neural network architectures without any prior knowledge demonstrates the feasibility of the neuro-evolution approach. There are several ways the architecture could be enhanced to make it more effective, including preprogrammed feature detectors and hierarchies of networks.

Since the networks are not given any prior knowledge about what features are relevant to playing go, they are forced to discover useful features themselves. Allowing the network to access a pre-defined feature space instead of looking at the raw board might make the task easier [2]. Such features could include common patterns and positions such as an eye or a group or even complicated constructs such as the ladder. These features would then be used as inputs to the neural network [1, 11]. It would still probably be useful to let the network develop its own features as well, but the pre-programmed features might allow it to learn faster and deal with more complex patterns.

SANE demonstrates the feasibility of evolving structures on more than one level at the same time. It should be possible to extend this idea and evolve a hierarchy of networks, where the lower levels would provide the inputs for networks at higher levels. In effect, evolution would be searching for an effective combination of networks, much the same way it is searching for an effective combination of weights and neurons now. When the task of playing go is broken into such subtasks, it may be the case that the number of generations required will increase with the number and size of the networks evolved and not with the size of the board. If this is the case, then evolving networks that play on full-size boards would no longer be computationally prohibitive.

9.3 Stronger Opponents

Even with more sophisticated architectures, stronger opponents are necessary in order to achieve truly high levels of play. The ability to use handicaps to simulate stronger opponents is a useful technique but not enough alone. The techniques used in handicap games are different than those that would be used against a stronger player in an even game. Handicap stones allow the weaker player to build stronger positions, but it still continues weak play from these positions. On the other hand, in an even game the opponent may play brilliant moves that would never be seen in a handicap game. If evolution is never exposed to such moves, it cannot develop comprehensive go skills.

A variety of stronger opponents would allow for a greater generality of play to evolve. However, it is not known how great the difference in play would be nor what the effect on the time required to evolve the networks would be. It is also not yet clear how much diversity is necessary to achieve general play.

One problem with using stronger opponents is that they tend to take considerably longer to generate moves than weaker programs. Given the large number of trial games generated every generation, it may not be possible to evolve against a slow opponent in a reasonable amount of time. However, the evaluation function might be modified to compensate for the lack of time. Each generation, a significant number of the networks evolved are significantly weaker than the networks of the previous generation. It should be possible to distinguish the weaker networks from the stronger networks by the use of a faster but weaker opponent. Only those networks that appear promising need be evaluated fully against the slower opponent.

Even if stronger computer opponents are used, eventually they would be exhausted. It would be necessary to find a way to evolve networks against actual human players. Given the popularity of internet-based go servers, there is no shortage of human players. However, there would be difficulties, particularly in the generation of fitness values. Fitness is used to distinguish the better networks from the worse networks in any given generation. It requires that the evaluation function be consistent for all networks evaluated. Since it would be unlikely for many different networks from the same generation to play the same human opponent, it would be difficult to assign a fair fitness value. The problem is compounded in that the strength of the human opponent is not always known and cannot be reliably used to weight game results against the strength of the human opponent. Nevertheless, good results have been reported in neuro-evolution with noisy evaluation functions[8], suggesting that the problems could be overcome. This way perhaps go-playing programs could finally be evolved that were able to compete with the best humans.

10 Conclusions

Traditional artificial intelligence techniques have been insufficient for building go programs that would be competitive at high levels of play. It appears new techniques based on pattern recognition and learning will be required to reach these levels. The SANE neuro-evolution approach is one such promising direction. Networks were evolved to defeat a publicly available go program on small boards with no pre-programmed knowledge of the game, and they exhibited several aspects of general go strategies.

References

[1] Herbert D. Enderton. The Golem go program. Technical Report CMU-CS-92-101, School of Computer Science, Carnegie Mellon University, 1991.

[2] Markus Enzenberger. The integration of a priori knowledge into a go playing neural network. Manuscript, 1996.

[3] Faustino Gomez and Risto Miikkulainen. Incremental evolution of complex general behavior. *Adaptive Behavior*, 5:317–342, 1997.

[4] Feng hsiung Hsu, Thomas Anantharaman, Murray Campbell, and Andreas Nowatzyk. A grandmaster chess machine. *Scientific American*, 263:44–50, 1990.

[5] Kai-Fu Lee and S. Mahajan. The development of a world-class othello program. *Artificial Intelligence*, 43:21–36, 1990.

[6] David Moriarty and Risto Miikkulainen. Discovering complex Othello strategies through evolutionary neural networks. *Connection Science*, 7:195–209, 1995.

[7] David Moriarty and Risto Miikkulainen. Efficient reinforcement learning through symbiotic evolution. *Machine Learning*, 22:11–32, 1996.

[8] David E. Moriarty. *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*. PhD thesis, Department of Computer Sciences, The University of Texas at Austin, 1997. Technical Report UT-AI97-259.

[9] David E. Moriarty and Risto Miikkulainen. Evolving obstacle avoidance behavior in a robot arm. In P. Maes, M. Mataric, J.-A. Meyer, and J. Pollack, editors, *From Animals to Animats: The Fourth International Conference on Simulation of Adaptive Behavior (SAB96)*, 1996.

[10] David E. Moriarty and Risto Miikkulainen. Learning sequential decision tasks. In M. J. Patel and V. Honavar, editors, *Evolutionary Synthesis of Neural Systems*. MIT-Press, Cambridge, MA, in press.

[11] Barney Pell. Exploratory learning in the game of go. In D.N.L. Levy and D.F. Beal, editors, *Heuristic Programming in Artificial Intelligence 2 - The Second Computer Olympiad*. Ellis Horwood, 1991.

[12] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[13] Jonathan Schaeffer, Robert Lake, Paul Lu, and Martin Bryant. CHINOOK: The world man-machine checkers champion. *The AI Magazine*, 16(1):21–29, 1996.