Applying ESP and Region Specialists to Neuro-Evolution for Go

Andrés Santiago Pérez-Bergquist
The University of Texas at Austin
aspb@mapache.org

3 May 2001

Abstract

Go is one notable board game where computer competence still trails behind that of human experts. In the past, neural-network-based approaches have shown promise. In this paper, the ESP variant of the SANE neuro-evolution algorithm was applied to go, and an alternate network architecture featuring subnetworks specialized for certain board regions was implemented. ESP produced simpler networks that performed just as well as the more complex ones produced by SANE in other studies. Having region-specialist subnetworks improved the already great performance marginally. However, both the simple network and the network with specialists failed to scale up to board sizes larger than 7x7.

1. Introduction

While most board games have fallen to computer opponents, go remains unconquered as of now. Though its rules are almost trivial, its explosive complexity of play has prevented it from succumbing to the sort of brute-force search that has proven effective in chess. Probably because go relies greatly on pattern and shape, approaches using neural networks have shown success in the past, especially on smaller boards [4, 5, 10]. This paper first seeks to see how well the ESP neuro-evolution algorithm works at developing computer go opponents, because this particular technique has not been tried previously. Then, it explores ways of improving performance by forcing parts of the network to specialize on different areas of the board, and briefly touches on how reusable such location-targeted expertise is.

The paper begins with a brief overview of go and of neuro-evolution techniques, explains the network architectures used and the tests run, then discusses the results and points out directions in which the work may be extended in the future.

2. Go

Go is an ancient board game originating in east Asia which has gained popularity worldwide. The rules are incredibly simple, and can be learned almost instantly, but becoming an accomplished player with effective strategies takes most of a lifetime. Unlike games such as chess, it remains a largely unconquered and computationally interesting problem in that the best artificial opponents now available, while capable of defeating the average player, are far below the skill of world champions.

2.1 The Rules of Go

Go is for two players, who face off on a board of 19 by 19 lines. Stones are played on the intersections of these lines, with the first player placing black stones and the second one white. A turn consists of placing a single stone at an empty intersection, or passing.

A contiguous set of like-colored stones is known as a group. (Adjacency counts only along the lines of the board, not diagonals.) The number of empty intersections adjacent to a group is known as the group's liberties. A group of stones with no liberties is immediately removed from the board. In the case where a move fills in the last liberty of at least one group of each color, the group(s) belonging to the player who did not just play are removed; the group of the player who did play now has liberties where the opposing stones were removed, and stays. Under modern rules, suicide via filling in the last liberty of a group you control with a move that captures nothing is not allowed, although there would be no benefit to so doing.

The *ko* rule prevents one from playing a move that exactly undoes the opponent's last move. (This is possible where a single stone has just been captured, and the capturing stone could be itself captured by replacing the captured stone.) In modern rules, this has been extended to the super-ko rule prohibiting the exact same position on the entire board twice in the same game.

The game ends after two consecutive passes. (In some rules variants with simple ko, a pass does not count towards ending the game if there was a move that was unavailable due to ko, as that move is now available after the double pass. This is relevant, as the opponent used in the research implemented such a ruleset.)

Scoring at the end of the game is based on the amount of territory controlled. Under Chinese scoring, each player received points equal to the number of intersections either under stones of his color or surrounded exclusively by stones of his color. Under Japanese scoring, each player receives points equal to the number of intersections surrounded exclusively by stones of his color plus the number of enemy stones captured. If neither player passed until the end of the game, then both scoring methods give the same difference between the two player's scores, because every enemy stone captured reduces the number of stones he has left on the board. In addition, the white player, who always goes second, receives a number of bonus points known as the *komi* to make up for going second. On a 19x19 board, this is 5.5. On all board sizes, the komi is always one-half more than an integer, preventing ties.

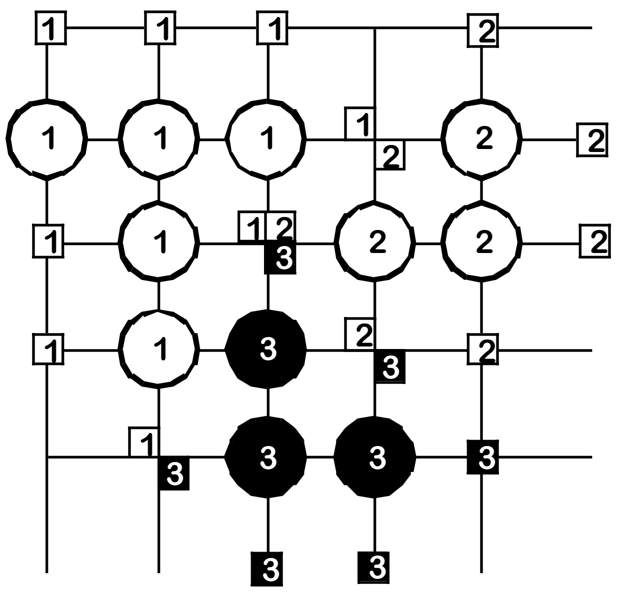


Figure 1. All stones with the same number belong to a single group. The liberties of each group are marked by the boxed numbers.

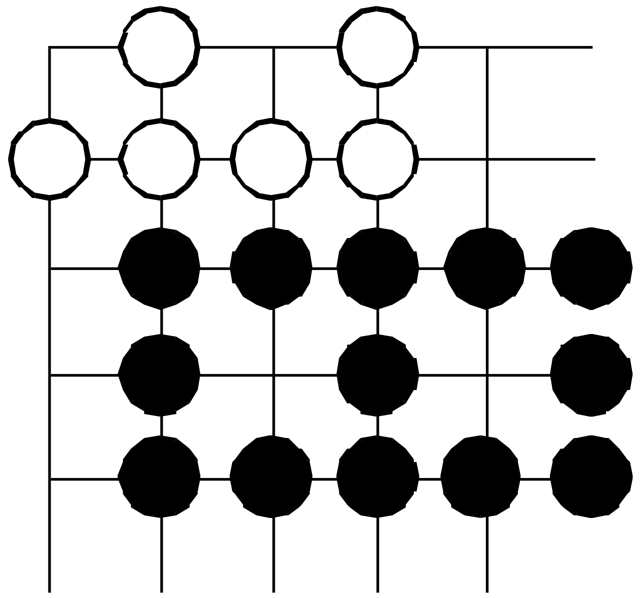


Figure 2. Both the white group and the black group have two disjoint internal liberties, or *eyes*, and can never be captured.

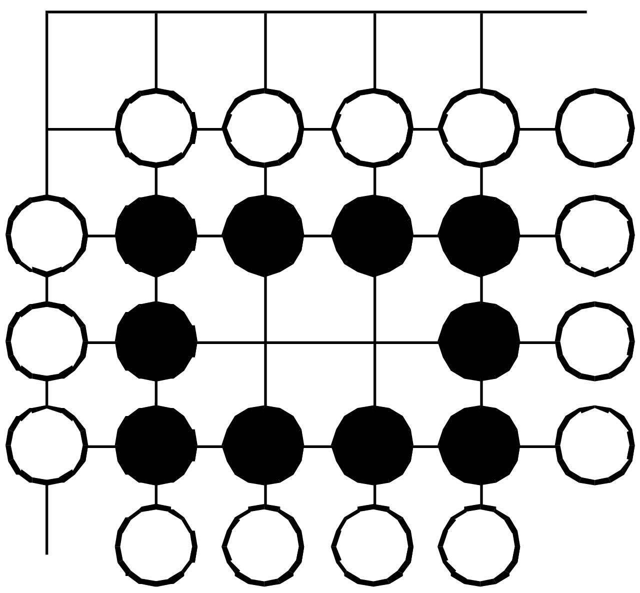


Figure 3. Although the black group has two liberties, it only has one eye, and can be captured by white.

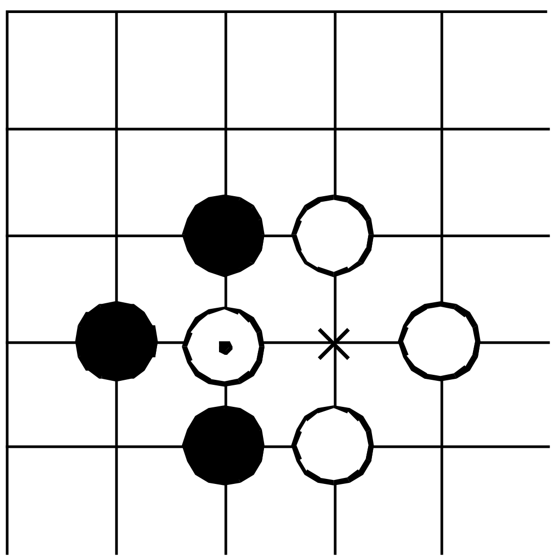


Figure 4. A ko situation–if black plays to the X, then white cannot immediately replay a stone to the position marked with the dot.

2.2 Computer Go

Several features of go combine to make it a harder problem for computers than most other popular board games, such as chess [12]. A regular go board has 361 intersections, and thus that many possible moves at the beginning of the game. Even by the end of the game, there typically remains a significant fraction of empty space on the board. Second, games of go are long in terms of moves. A typical grand-master-level chess game lasts thirty to forty moves, but go can take over two hundred. Additionally, the consequences of a given move may not come into play for many dozen moves. Combined, these mean that the sort of brute-force search algorithms used in chess fail miserably, as the number of paths to explore increases much faster the further one looks ahead, and effective play requires looking ahead much further. (Deep Blue, current reigning world chess champion, looks ahead about 25 moves—almost to the end of the game even at the very beginning.) The disconnection between moves and consequences exacerbates the horizon problem, wherein the negative consequences of a play can be put off until just past the look-ahead horizon of the program.

Some attempts have been made to improve this otherwise dismal situation. Most go programs include standard opening libraries that take care of the initial *fuseki* or opening game [8]. Choosing among specific opening tactics is still rather touch and go, however. In the end game, most moves are fairly local, and options are limited, so brute force can help, but by this point the score is largely settled. The critical mid-game is where traditional programs are weakest.

Human go players, in describing their games, do not generally speak of analyzing particular moves in extended detail. Instead, much of the game-related terminology speaks of the shape of groups and the board as a whole. Formations are said to have "good shape" and "bad shape". Players rely strongly on pattern recognition and intuition about a situation, which makes it hard to codify their expertise. Neural networks are quite effective at pattern-recognition and detecting vague properties such as shape and form [2], so some work has been done in recent years concerning their application to go [4, 5, 10].

3. Neuro-evolution

Neural networks make use of networks of artificial neurons, which are processing units that compute some simple function of their input values, producing one or more output values. Typical neurons compute a weighted sum of their numerical inputs, then apply a threshold function or a sigmoidal function (if the output function must be differentiable). Networks as a whole are trained to perform a particular task by repeatedly attempting the task and providing corrective feedback.

Learning algorithms are best suited for when the correct way to perform a task is known. At each stage, the output of the network is compared with the desired output for that input, and adjustments are made to the weights of individual neurons to reduce the error. Over many iterations of this process, the network learns the desired function.

Go does not fit conveniently into such a paradigm, as we do not know what the correct move at a given point in a game is. Even classifying a particular move as beneficial or not is difficult, as the consequences of a move are far-reaching, and disentangling them from consequences of other moves is difficult at best. About the only feedback we receive is the final scores of the game that determine the winner. In such a situation, we can apply evolution instead of learning.

In evolutionary techniques, the weights of a neuron are coded into a genome, and a large population of such genomes is produced. Then, we assemble many neural networks using randomly chosen neurons, observe their performance, and select those that did best. A new pool of candidate neurons is then produced by interpolating the genomes of the neurons, simulating reproduction, and adding in random variations, simulating mutation. Over the course of many generations, populations of well-performing neurons are produced. Due to its less-direct nature, evolution is generally more time-intensive than learning, much as in the real world.

3.1 SANE

In the Symbiotic Adaptive Neuro-Evolution algorithm, we deal with a population of individual neurons, each of which is represented by a numerical vector defining the weight of its connections to each of the input and output units [3, 4, 5, 6, 10]. In addition, there is a population of network blueprints consisting of pointers to neurons in the population. Individual networks are built out of the subset of the neurons specified by one of the blueprints, and the performance of the network as a whole is assigned to both the blueprint and to each individual neuron contributing to the network. The performance of a neuron is averaged to compensate for different neurons being members of varying numbers of networks.

After each generation has been evaluated, the genomes of the top performing neurons are recombined and mutated. In addition, the network blueprints are also evolved, with crossover recombination between network representations, mutations where half of the pointers are changed to offspring of the neurons to which they previously pointed, and mutations where a small fraction of the pointers are set to completely random neurons. Similarly, the top-performing neurons themselves are also recombined and mutated to produce new neurons.

The use of blueprints ensures some level of consistency in network composition from generation to generation, and helps to prevent high levels of redundancy where multiple identical or nearly-identical neurons are found in a network. Even without blueprints, there is still a tendency to preserve diversity in the neuron population, as high-performing networks require multiple sorts of neurons. However, SANE does have a few problems. Many times, neurons with completely different weights and purposes are bred, producing offspring that fill no useful role, although sometimes this creates a new neuron type that is indeed needed. Achieving a critical mass of a given neuron type, wherein such neurons breed with reasonably similar neurons often enough to produce useful results, can be difficult. The obvious solution is to segregate neurons on the basis of type.

3.2 ESP

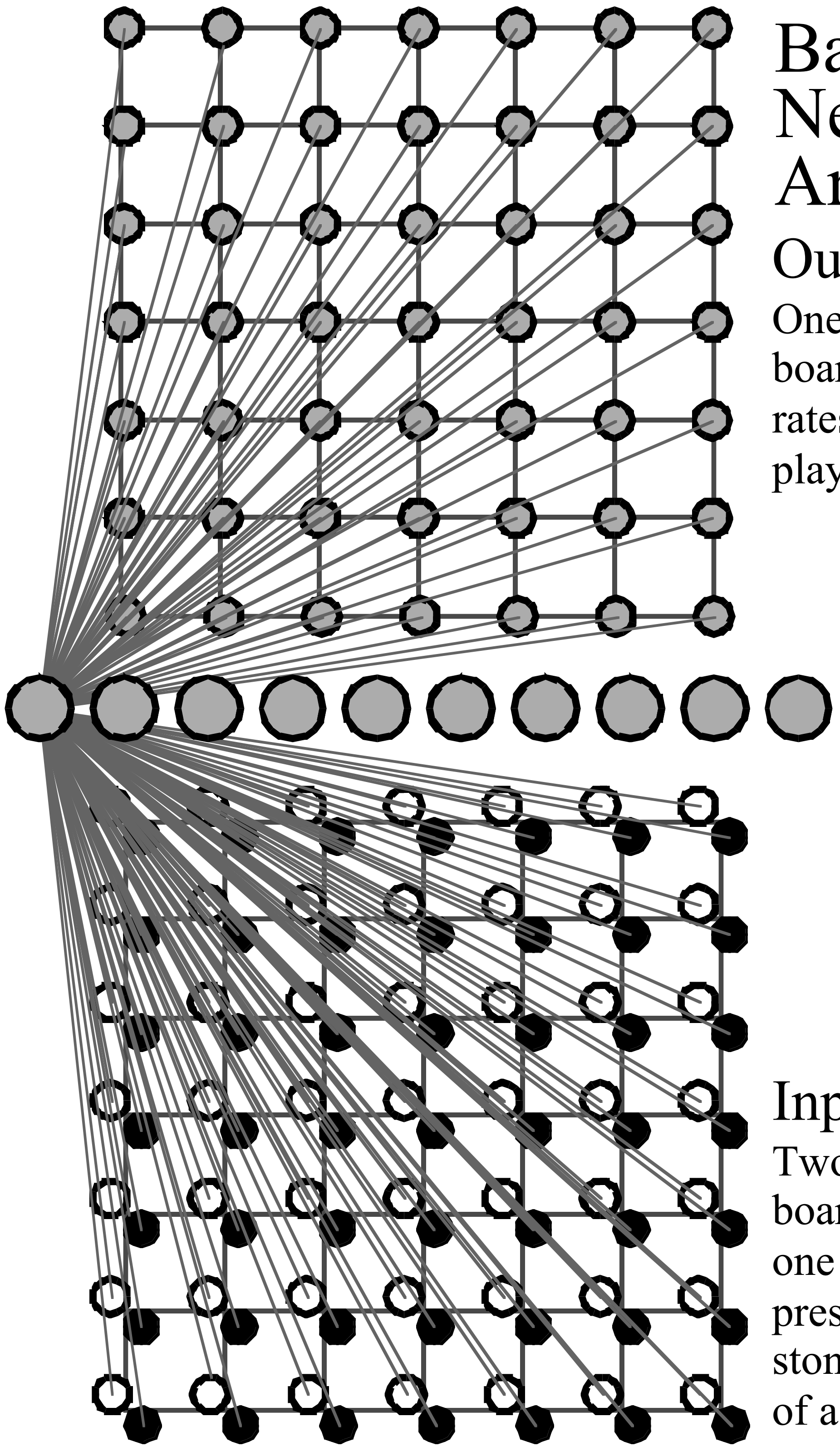
The Enforced Sub-Population variant of SANE is based on just this sort of separation [3, 6]. Rather than having one large pool of neurons with network blueprints, ESP maintains a separate population of neurons for each position in the network. Building a network then consists of selecting exactly one neuron from each of these sub-populations. Since the populations are kept separate during the creation of the next generation, each population is able to focus on a particular function more quickly, and networks are less likely to have redundant neurons. ESP was used as the mechanism of evolution for this research.

4. Network Architecture & Experimental Design

The initial network architecture was created to establish a baseline for how well ESP could evolve networks for defeating an algorithmic opponent at various board sizes. This architecture was scalable to different board sizes. It had $2n^2+1$ input units and n^2 output units, where n is the length of the edge of the board. The first $2n^2$ input units correspond two each to a board position, with the first one reading 1 if that position is occupied by a white stone, zero otherwise, and the second reading 1 if that position is occupied by a black stone, zero otherwise. (Two sensors per position were chosen, as opposed to some one-sensor scheme such as -1 = black, 0 = empty, 1 = white, because it was felt that occupation by black and by white were really distinct concepts, not ends of a continuum with "no stone" in the middle, and neural networks tend to respond poorly to input codings that do not correspond to direct shades of meaning.) The last input was an integer giving the number of net stones captured by the network over the course of the game; this is an important piece of information for scoring purposes, and the network had no other way of knowing this, since it was a simple feed-forward network with no memory of previous states in the game. No explicit representation of the topology of the board (i.e. the proximity of one location to another) was provided.

The n^2 output units correspond to each of the positions on the board. The network was run by giving it the current board position as input, and the value at each output unit was

Basic Network Architecture



Output Layer

One unit per board position; rates desirability of playing there.

Hidden Layer

Each neuron is connected to each input and output unit (only one set of connections is shown, for clarity); weights of connections are encoded in genes.

Input Layer

Two units per board position; one detects the presence of a white stone, and the other of a black stone.

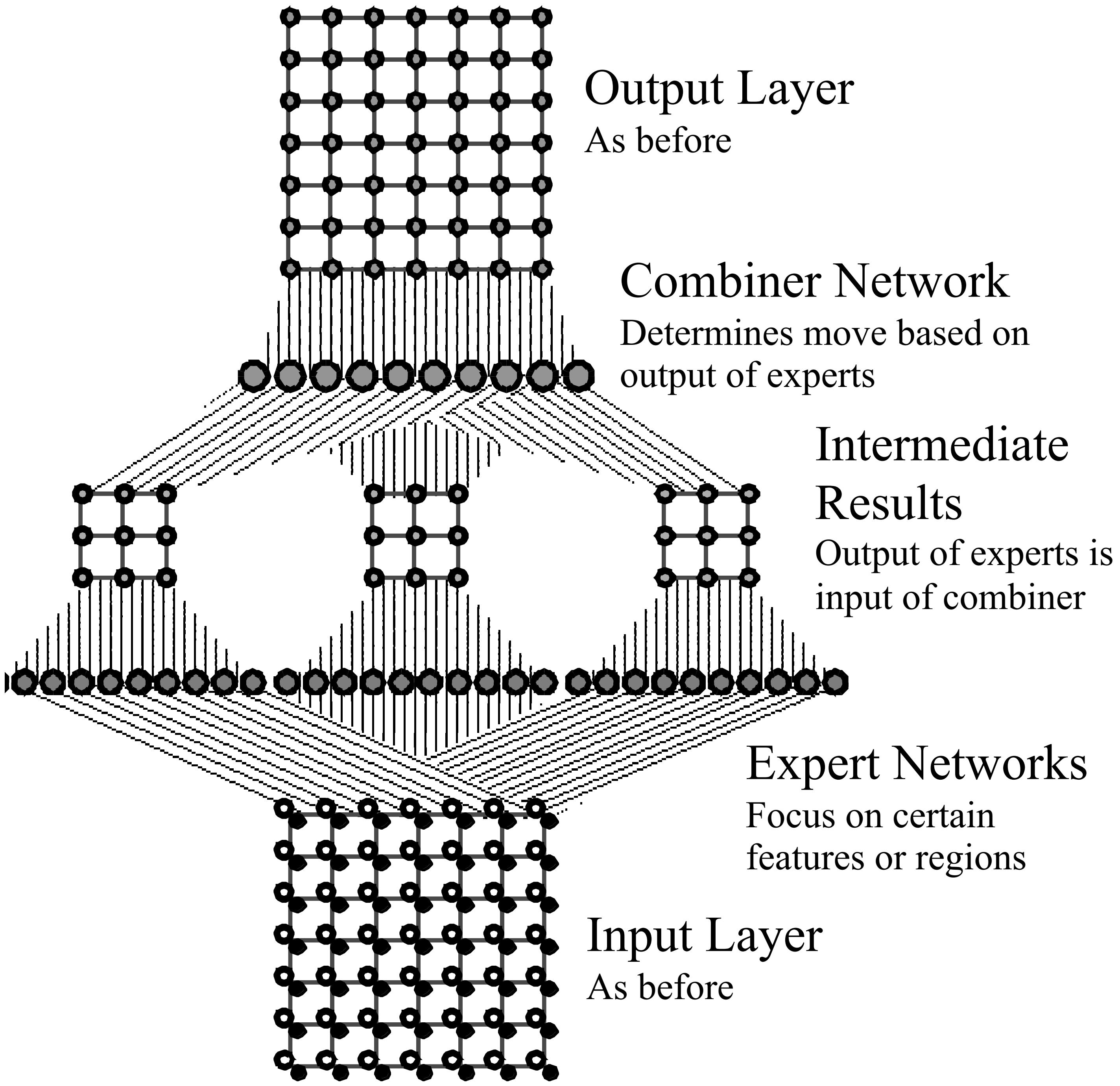
interpreted as how much the network would like to play at that location next. The highest-rated legal move was then played. (The network was not expected to learn the actual rules of the game, only rate moves. Making an illegal move was not possible.) If the desirability of all legal moves fell below a certain threshold, then no move was made, and a pass was played. The opponent was then allowed to reply, and the entire process repeated.

The opponent was GnuGo, the best available open-source computer go player. GnuGo is by nature deterministic. For a given board position, it always makes the same move. Faced with a deterministic opponent, the neural network can learn the exact sequence of moves that will beat it, without developing broader strategy. To counteract this, GnuGo was modified to make a completely random move some fraction of the time. This would throw it out of its deterministic pattern without affecting its overall skill too much, making it a better opponent for training against.

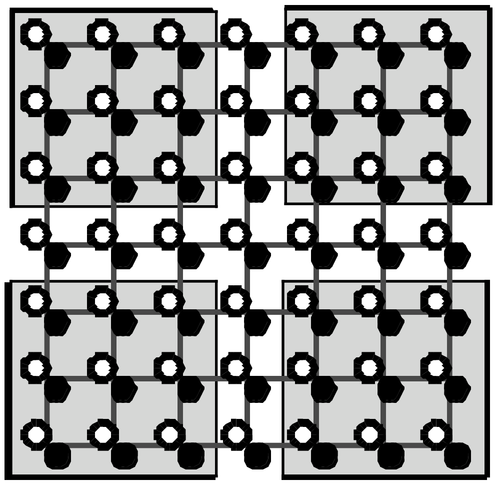
Afterwards, a two-layer architecture was created in effort to improve performance at a given size, as well as test how well knowledge gained from one board size generalized to others. The overall inputs and outputs were as before. However, the inputs from the board went to one of three special networks. Each looked at a 3x3 portion of the board, with one trained to look at corners, one at edges, and one at areas in the middle of the board. (The inputs for corner and edge networks were rotated to a common orientation.) The board was tiled by these 3x3 subareas, with an overlap of one position between adjacent tiles. Thus, a 7x7 network had nine tiles and a 9x9 network had 16. Each of these networks was used as many times as needed; that is, each corner was looked at by an identical corner-specialist, and so on. Each of these subnets had nine output units, for some intermediate evaluation of the 3x3 area under consideration. All the outputs from these specialist networks, as well the one unit encoding net captures, were fed as inputs to a network with nxn outputs which encoded the desirability of moves as before.

The performance of these two-layer networks was compared to that of the single-layer networks. Additionally, trials were made where previously trained front networks were fixed and no longer allowed to evolve, and a new combiner network was evolved, in an attempt to see how much useful information the front region-specialist networks encoded. This was done with front networks trained on a 7x7 board, then reused on both a 7x7 and a 9x9 board.

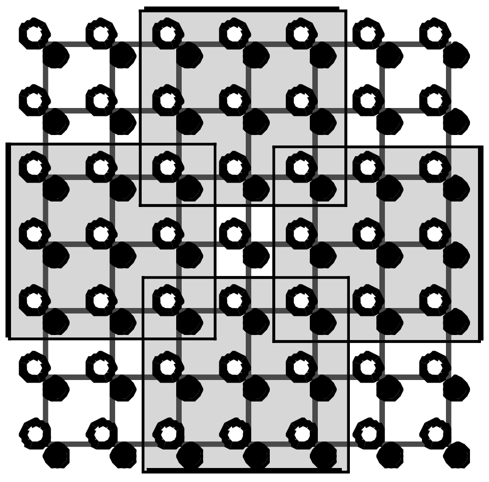
Layered Network Architecture


Output Layer
As before
Combiner Network
Determines move based on output of experts
Intermediate Results
Output of experts is input of combiner
Expert Networks
Focus on certain features or regions
Input Layer
As before

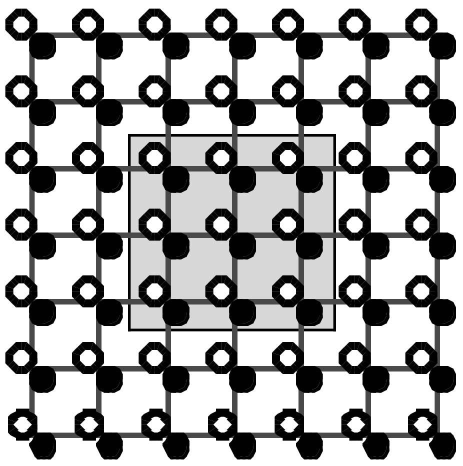
Region Specialists on 7x7



Corner Networks

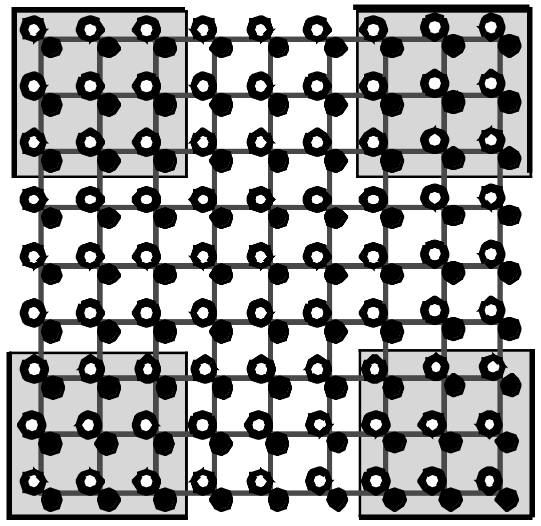


Edge Networks

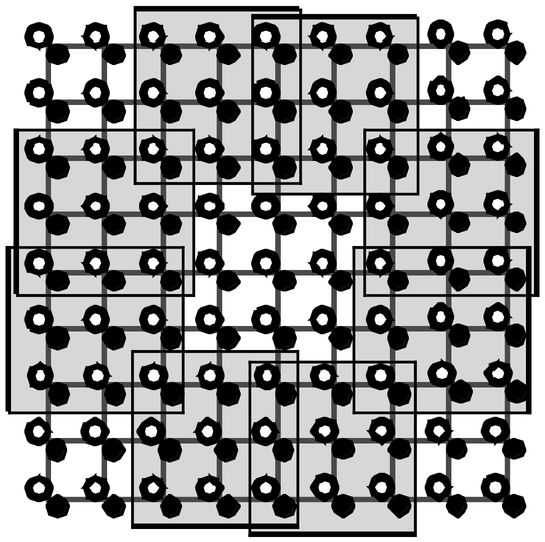


Center Network

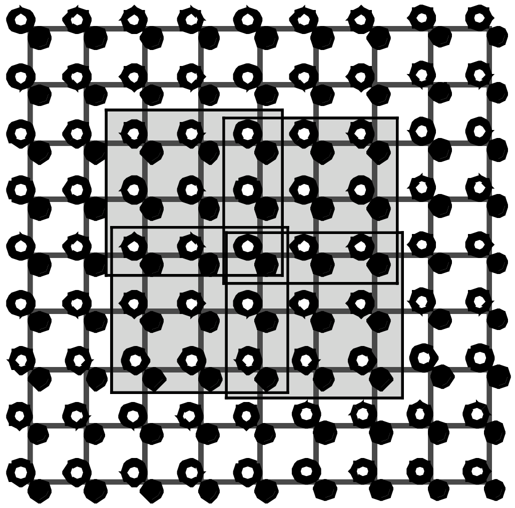
Region Specialists on 9x9



Corner Networks



Edge Networks



Center Networks

5. Trials & Results

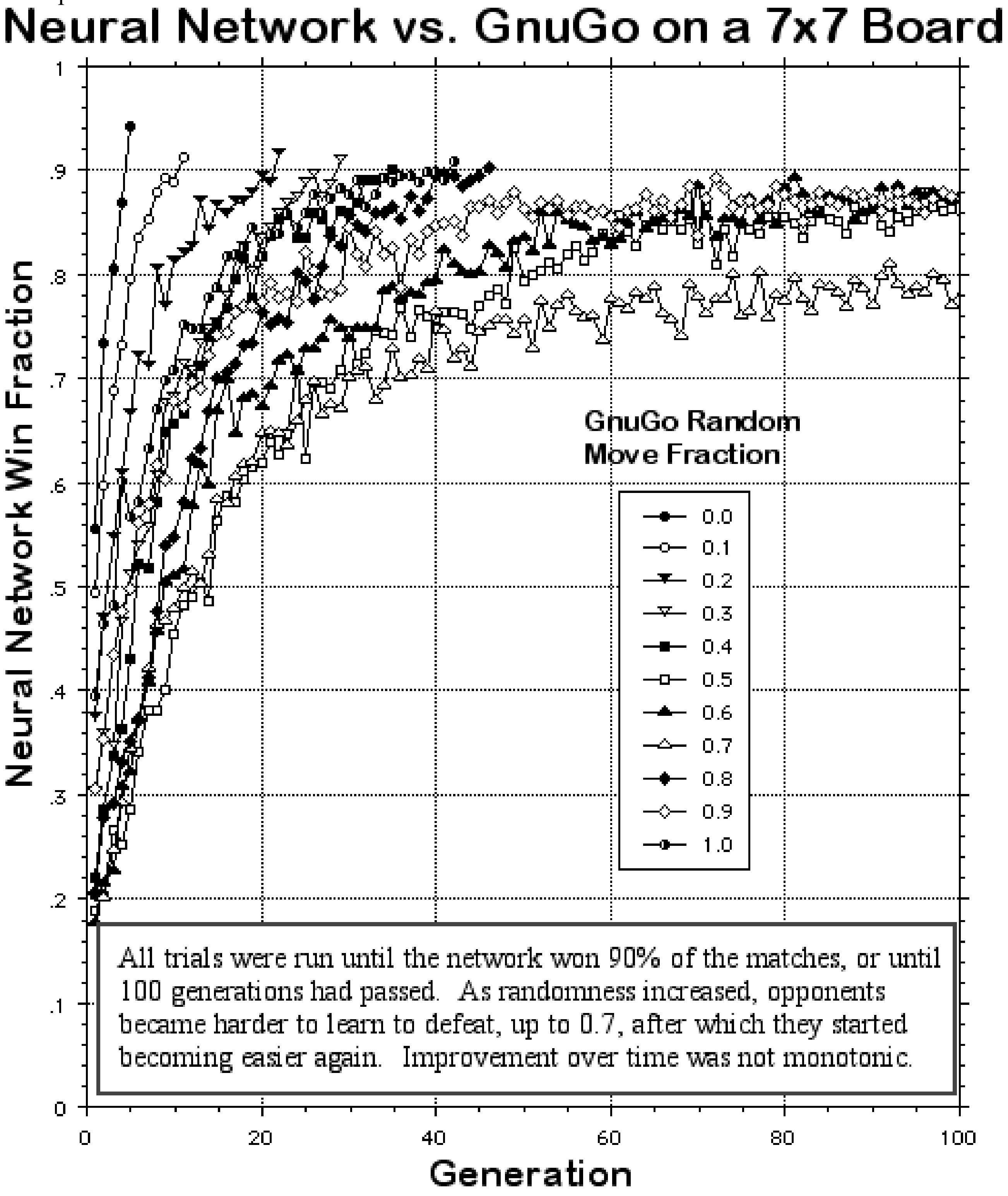
Initial runs were done with a completely deterministic GnuGo on boards of size 3x3 to debug the code, with some trials at 5x5 and 7x7 to ensure some level of scalability. Based on the runtimes experienced even at 7x7, larger boards were not attempted until later in the project when final results were desired.

A series of trials was run with the random move fraction of the GnuGo opponent set at each value from 0 to 1 in increments of 0.1, at board sizes of 3x3, 5x5, and 7x7. In general, the difficulty of learning to defeat the opponent rose as the randomness increased up to 0.7, after which it became easier again. On the basis of these results, it was determined to use a randomness fraction of 0.3 throughout the rest of the work. While most other researchers have previously used 0.1, at that level of randomness on small board sizes it is still reasonably probable to have games without any random moves, so that a network that beat only that one game could still score fairly highly.

Trials were then attempted on boards of size 9x9, 11x11, and 13x13. In every single case, no network was ever evolved that won more that 0.012 of the games in a given generation. There was no trend towards improvement, with the fraction of games won hovering mostly in the 0.001 to 0.005 range, even after 200 generations. This is in stark contrast to smaller board sizes, where the network would be typically be winning 0.75 of the games within 10 generations and 0.85 to 0.90 of the games within 20 generations.

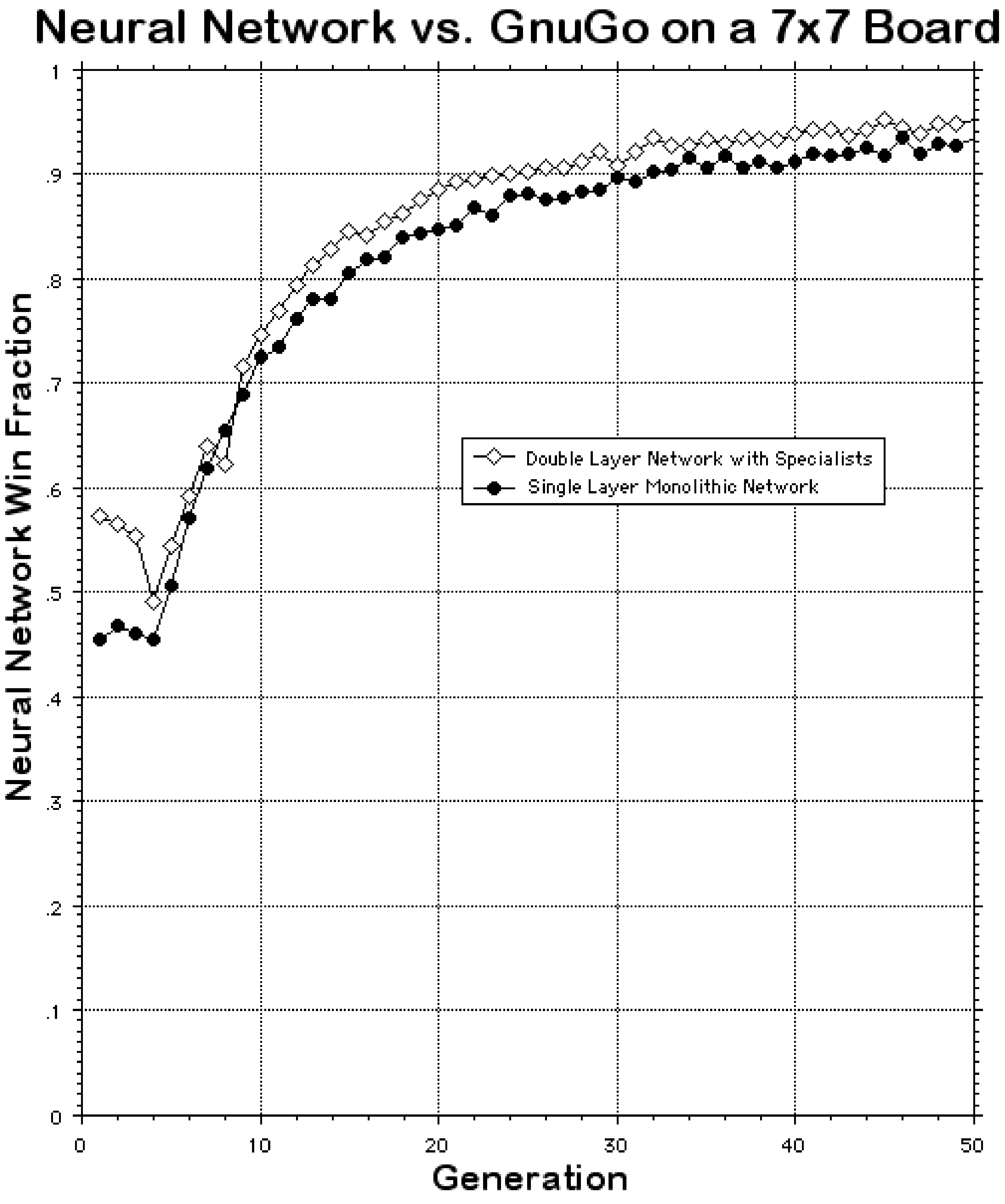
All trials up to this point had been with networks that initially had 10 hidden-layer neurons, which would be dynamically adjusted if needed. (However, in no case did the ESP algorithm determine that the network performance had stagnated and that an additional neuron would improve performance.) It was theorized that the complexity of the game might be exceeding the ability of ten neurons to handle it at 9x9, so several trials were run with an initial 20 neurons. It was felt that since 10 neurons were more than adequate for 7x7, and 9x9 had less than twice as many board positions, doubling the hidden layer size should definitely provide enough complexity to handle the problem if mere network size was indeed the stumbling block.

Unfortunately, there was no impact on performance whatsoever; it remained essentially zero wins plus some noise.



Next, several trials were done with both single-layer networks and double-layer networks on a 7x7 board. Since the single-layer networks were already evolving to the point that they could beat GnuGo nearly every time, there was not much to be expected in the way of improving peak performance, but the number of generations needed to achieve this performance might be less. In fact, the double-layer networks on average slightly outperformed the single-layer networks not just initially, but asymptotically as well, as shown on the next graph depicting the average performance of the two types of networks over six runs. A paired t-test confirmed that the small observed difference between the double-layer and single-layer networks was indeed statistically significant, with $P < .0001$. Double-layer networks were also tried at 9x9, with no success.

Finally, the front region-specific networks of the best double-layer network for 7x7 boards were taken and fixed, and trials were run with new 9x9 networks where only the rear combiner network was evolved, in an effort to see if the information from the front networks was of any use by itself and would generalize to higher board sizes. Like all previous attempts on the 9x9 boards, no strategy whatsoever emerged. As a control, the same test was tried on 7x7 boards, and the average performance was indistinguishable from that of the networks which were started entirely from scratch.


Neural Network vs. GnuGo on a 7x7 Board
Neural Network Win Fraction
Generation
Double Layer Network with Specialists
Single Layer Monolithic Network
0
.1
.2
.3
.4
.5
.6
.7
.8
.9
1
0
10
20
30
40
50

6. Discussion

The clearest result by far is that none of the techniques tried so far work at board sizes of 9x9 and up. Some sort of change takes place there. Whatever the aspect of the game that gets more complicated, simply throwing more neurons at it does not appear to be a solution. A reasonable explanation is that at small board sizes, a random evolutionary search technique that relies only on end-of-game scores can effectively explore the game space, but that at 9x9 the game becomes sufficiently complex that the evolutionary search cannot even find a hill to climb. This may be related to comments made in [9] that go is not really go at board sizes less than 7x7. Certainly at 3x3, the best move is often passing, as the board is so crowded no real groups can exist. Subjectively, 7x7 or 9x9 is where the game starts to feel interesting. The GnuGo documentation makes note that GnuGo is optimized for larger board sizes, and may perform poorly on small boards. This could also be helping the network. Still, the author, who, although an avid game-player, has no prior go experience, had no trouble defeating GnuGo even at 9x9 and 11x11. Review of game logs reinforces the dim view of GnuGo's abilities, revealing many cases where GnuGo will fill in an eye of one of its groups rather than pass, sometimes costing it the game. The network does not play much better, which is not surprising given that it learned from GnuGo.

The second main point noted is that the double-layer network was more effective than the single-layer network. While the difference was small, there was not too much room for improvement over the single-layer network. Of particular interest is that the double-layer network had notably better performance towards the beginning, then dropped to nearly the level of the single-layer network, and remained somewhat above it thereafter.

Trials involving the reuse of the front networks indicated that by themselves, they are fairly useless. This was somewhat expected, given that the actual move made is chosen by the back network, which was restarted from a random population. The only way that the front networks could have helped was by keeping their intermediate evaluation function stable over time, giving the rear network a stationary target in terms of the meanings of its inputs. That this approach still failed at 9x9 was also not surprising, since the complexity of the back network, which has to understand the board as a whole, is almost that of a single-layer network for the same board size.

In previous work [5, 10] applying SANE to go, it was found that networks with about 300 neurons were needed for optimal performance on a 7x7 board. ESP did fine with a mere 10 neurons per network. Certainly, for this task, ESP finds a much more efficient encoding. This is probably accounted for by the SANE networks have large amounts of redundancy in the form of fairly large numbers of identical or near-identical neurons. Since each neuron in an ESP network comes from a completely separate population, there is no reason that distinct populations would have any similarities in their member neurons.

7. Future Work

Several areas remain to be explored in future research on this topic. Some of them were considered as possible directions in which to take this project, but were not pursued due to a need to limit the research's scope because of time constraints. Others arose naturally as questions when reviewing the results obtained.

Developing two-layer networks where the front layer specializes in some manner other than by areas of the board was one of the clearest extensions to pursue. Some work along these lines has already been done in [4] using NSANE, which is essentially SANE with multiple subnetworks evolved from completely separate neuron and blueprint populations. It was found to have no measurable effect compared to a single-layer network, possibly because there was no incentive for the front networks to diversify from each other. In the case of area-specific networks, each receives different input, so this was not an issue in the work done. If using multiple front networks that each looked at the entire board, one might need some sort of stimulus in the evolution function that rewarded some level of diversity, possibly as measured by correlation between the outputs of the networks. Then again, any added performance boosts that diversity might bring about could be enough to cause it to arise on its own [1], at least in the dynamics of a system run by ESP rather NSANE.

One clear limiting factor, at least on board sizes up to 7x7, was the skill of GnuGo. Since even the first networks tried were learning to beat GnuGo over 90% of the time within twenty generations, it seems clear that GnuGo was not posing much of a challenge. Unfortunately, few alternatives exist. While there are superior computer opponents for go, none of the others found were open-source research projects, but rather proprietary for-profit endeavors. Using human players springs to mind, and this could be facilitated by the use of one of the several go servers on the internet. The program could be designed to connect to these telnet-based servers and interface with them the same way a human does, challenging anyone willing to take it up. The problem is that even with the potentially large pool of players thus available, one would still not be able to play nearly as many games as quickly as was done in the research. Each generation consisted of 10,000 games, and there are at most a few dozen go players on each of the two or three major go servers at any given time. Still, human opponents could be combined with a computer opponent, yield both new challenging opponents as well as large amounts of games.

The other alternative is to make use of competitive co-evolution, allowing the network to play either itself or another population of networks [7, 11]. In such an approach, the network could be primed by evolving against GnuGo for a while, to a develop some sense of how to play go; then once GnuGo was routinely beaten, the network could be played against other networks from the same population, or from another population of parasites bred to attempt to trip up the main, or host, network. Such a host-parasite distinction is maintained by penalizing hosts severely for losing, while rewarding parasites highly for winning. This encourages the hosts to become resistant to any sort of attack, and the parasites to concentrate on simpler tricks that need only work occasionally. Once the network became sufficiently good, the parasites themselves would probably evolve to become reasonably good general-purpose players themselves. In such an approach, techniques are required to ensure that both populations do not wander off into some corner of strategy space that defeats its current opponents, but is actually really bad against a normal opponent. This can be done either by occasionally playing against older champion networks or against GnuGo, so that the networks do not lose the ability to defeat opponents they once could. Some work was done with co-evolution under SANE on 7x7 boards in [4].

One additional problem is that all the network architectures used, both in this work and in the references consulted, are board-size-specific. The focus of an entire project could be developing an architecture that works on multiple board sizes without retraining. Attempting to reuse the 7x7 front networks on a 9x9 board is a step towards size-independence, but not a very good one, as the key back network is still good for only the board size on which it was trained. A successful design that was size-independent would probably feature several networks that focus on some large sections of the board and could be slid around with varying degrees of overlap, with some sort of method for selecting between their outputs.

8. Conclusions

While neuro-evolution had been previously applied to go, this was the first known application of the ESP evolution algorithm. It proved quite effective at small board sizes, exhibiting excellent performance with networks an order of magnitude less complex than those needed by SANE. This success did not scale, however, with a sudden and total dropoff in performance at boards of size 9x9 and above, even though SANE had previously shown a more gradual decrease in performance at larger sizes.

Multiple layer networks with some sort of in-built structure outperformed monolithic networks with no explicit differentiation, but by a small margin. This approach still did not scale, however. Additionally, the amount of reusable information in a network was deemed fairly low, indicating that this is a good approach for solving very specific problems, but it might not be as well suited for situations requiring generalization or an internal process that humans can observe and identify.

9. Acknowledgments

The author would like to thank Risto Miikkulainen, for his help, support, and guidance as advisor to this research project; Ray Mooney, for being the second reader of this thesis; Mohamed Gouda, the departmental honors program supervisor; and Faustino Gomez, for creating and making available the general-purpose ESP code.

10. References

[1] Tucker Balch. Learning Roles: Behavioral Diversity in Robot Teams. *AAAI Workshop on Multiagent Learning,* 1997.

[2] Kumar Chellapilla & David B. Fogel. Evolution, Neural Networks, Games, and Intelligence. *Proceedings of the IEEE.* Vol 87, No 9, September 1999.

[3] Faustino Gomez & Risto Miikkulainen. Incremental Evolution of Complex General Behavior. Manuscript, 1996.

[4] Todd Greer. Evolution of Hierarchal Neural Networks for Go. Manuscript, 1998.

[5] Alex Lubberts. Go in SANE. Manuscript, 2000.

[6] Daniel Polani & Risto Miikkulainen. Fast Reinforcement Learning through Eugenic Neuro-Evolution. *Technical Report AI 99-277,* 1999.

[7] Jordan B. Pollack, Alan D. Blair, & Mark Land. Coevolution of a Backgammon Player. *Fifth Artificial Life Conference*, 1996.

[8] Kiyoshi Kosugi & James Davies. *38 Basic Joseki.* Tokyo: The Ishi Press, 1973.

[9] rec.games.go *Usenet Newsgroup.* Fall 2000 - Spring 2001.

[10] Norman Richards, David E. Moriarty, and Risto Miikkulainen. Evolving Neural networks to play go. *Applied Intelligence*, 1998.

[11] Christopher D. Rosin & Richard K. Belew. Methods for Competitive Co-evolution: Finding Opponents Worth Beating. Manuscript, Undated.

[12] N. Sanechika. "Go Generation" - A Go Playing System. *ICOT Technical Report TR-545.* 1990.

[13] Kano Yoshinori. *Graded Go Problems for Beginners.* Tokyo: The Nihon Ki-in, 1985.