Learning Opening Strategy in the Game of Go

Timothy Huang, Graeme Connell, and Bryan McQuade
Department of Mathematics and Computer Science
Middlebury College
Middlebury, VT 05753
{huang, gconnell, bmcquade}@middlebury.edu

Abstract

In this paper, we present an experimental methodology and results for a machine learning approach to learning opening strategy in the game of Go, a game for which the best computer programs play only at the level of an advanced beginning human player. While the evaluation function in most computer Go programs consists of a carefully crafted combination of pattern matchers, expert rules, and selective search, we employ a neural network trained by self-play using temporal difference learning. Our focus is on the sequence of moves made at the beginning of the game. Experimental results indicate that our approach is effective for learning opening strategy, and they also identify higher-level features of the game that improve the quality of the learned evaluation function.

Introduction

The ancient game of Go is one of the most widely played strategy games in the world, especially in the Far East. More effort has been devoted to developing computer programs that play Go than to developing programs for any other game of skill except chess (Müller 2000). Unlike chess, however, where the best programs play as well as the top human players, the best Go programs play only at the level of advanced beginners. Standard game-playing techniques based on brute force minimax search are not sufficient for Go because the game tree is extremely large and because accurate evaluation functions are slow and difficult to construct. Hence, most current programs rely on a carefully crafted combination of pattern matchers, expert rules, and selective search. Unfortunately, the engineering effort involved suggests that making significant progress by simply fine-tuning the individual components will become increasingly difficult and that additional approaches should be explored.

In this paper, we investigate a machine learning approach to constructing an evaluation function for Go based on training a neural network by self-play using temporal difference (TD) learning. We focus on learning opening strategy, i.e., the sequence of moves played at the beginning of the game. Also, we investigate a number of higher-level features of Go positions and identify those that most improve the quality of the evaluation function.

We begin by presenting an overview of game play in Go and the challenges for computer Go. After describing our learning approach and experimental methodology, we present and analyze experimental results. We conclude by surveying earlier work and discussing future plans.

Background

To start, we present a brief description of how Go is played by humans and by computers. A comprehensive explanation of the rules can be found in (Kim and Soo-hyun 1997); our aim here is simply to give a sense of the object of the game and the factors that make it difficult for computers to play.

Playing Go

Go is a two-player game played with black and white stones on a board of 19 x 19 intersecting lines, though 13 x 13 and 9 x 9 boards are sometimes used. Starting with black, the players take turns either placing one stone onto one of the empty intersections or passing. The goal is to acquire territory (empty intersections) by surrounding it with stones. Once placed, a stone does not move; however, blocks of stones can be surrounded and captured by the opposing player. The game ends when both players pass. The player who has surrounded the most points of territory wins the game. In official play, there are no draws because the white player receives 5.5 additional points (called *komi*) to compensate for playing second.

Stones placed horizontally or vertically adjacent to each other form a *block*. The empty intersections next to a block are its *liberties*. A block is captured when all of its liberties are occupied by enemy stones. Consider the following examples from Figure 1: In the bottom left corner, a block of white stones has surrounded 12 points of territory. Towards the bottom right, a block consisting of a single black stone is surrounded on three of four sides. White can capture this stone by playing at its last liberty, the intersection marked **A**. Above those stones, White can capture the block of two black stones by playing at the liberty marked **B**. If it is Black's turn, Black can help those stones escape by playing at **B** first and creating a resultant block with three liberties.

In the upper left corner, black can capture the white stones by playing at **C**, thereby fully surrounding the three stones. Normally, it is illegal to play a stone where it will not have any liberties. However, playing a black stone at **C**

is allowed because it captures at least one of the white stones directly adjacent to **C**. In this case, it captures the entire white block of three stones. In the upper right corner, the white block is *alive*, i.e., safe from capture, because black cannot play a stone at **D** and a stone at **E** simultaneously.

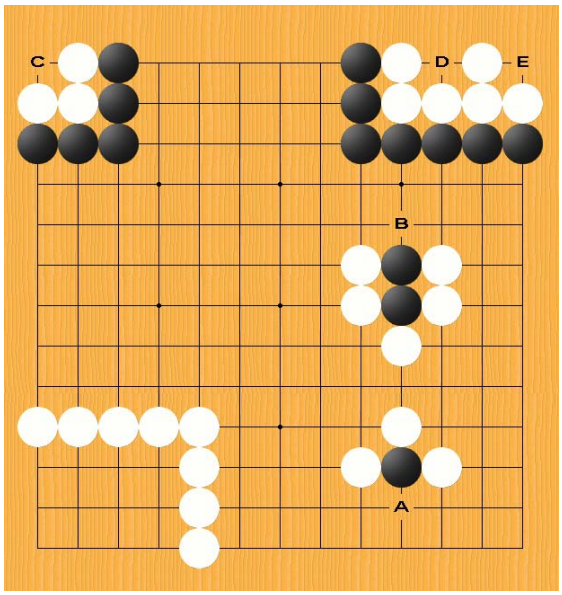

Figure 1: White can capture one black stone by playing at **A** or two stones by playing at **B**. Black can capture three white stones by playing at **C**. Since black cannot play at **D** and **E** simultaneously, the white stones in the upper right corner are alive.

The stones in Figure 1 were artificially arranged for explanatory purposes. Figure 2 shows a more realistic board situation partway through a game on a 13 **x** 13 board. In this game, white has staked out territory in the upper right, upper left, and lower left sections of the board. Meanwhile, black has staked out territory in the lower right and upper middle sections of the board and has pushed into the middle left section, thereby reducing white's territorial gain. Furthermore, black has effectively captured the white stones in the middle right section of the board (see (Kim and Soo-hyun 1997) for a full explanation).

Part of what makes Go so engrossing and challenging is the interplay between strategy and tactics. On one hand, players try to build stone patterns with "good shape", i.e., those with long-term strategic influence. On the other hand, they fight highly tactical "life-or-death" battles that concern whether a group of stones can be captured or not.

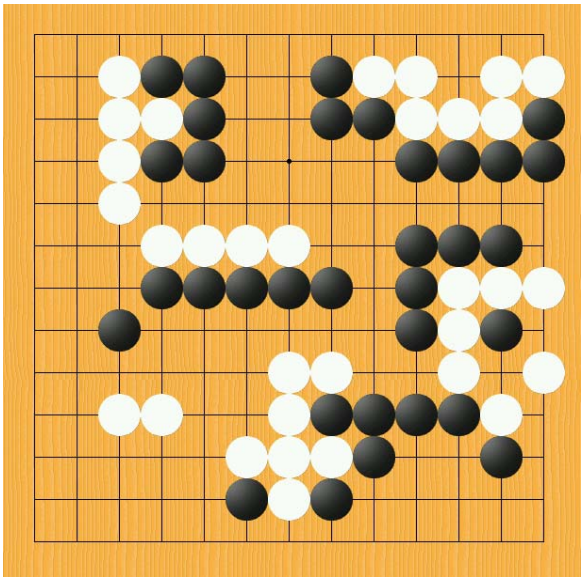

Figure 2: White's turn to move partway through an example game on a 13 **x** 13 board.

Figure 3 shows the board situation at the end of the same game from Figure 2. Black's territorial points are marked with small black dots, and white's territorial points are marked with small white dots. The final score, taking into account captured stones and komi, has white ahead by 4.5 points of territory.

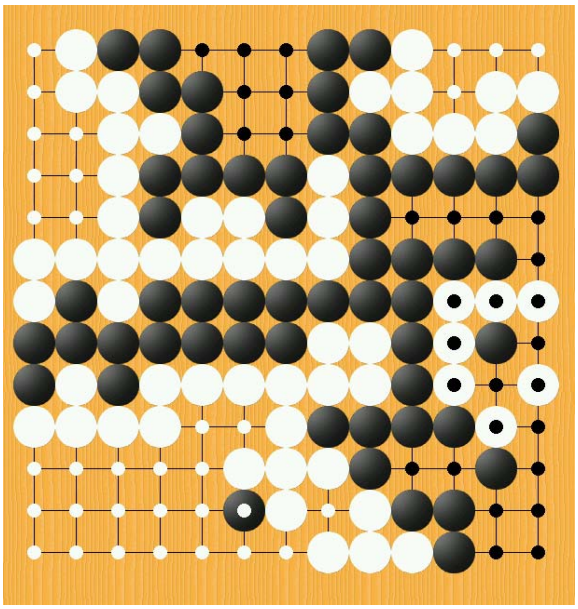

Figure 3: Final board position. Taking into account captured stones and komi, white wins by 4.5 points.

Computer Go

From a computer scientist's perspective, what stands out about Go is that it remains a perplexing computational

problem. The standard brute force minimax approach to strategy games is not sufficient for Go because the game tree is extremely large – the average branching factor is about 35 in chess and 250 in Go – and because accurate heuristic evaluation functions are computationally expensive.

Hence, most Go programs contain a mix of various AI components. Pattern matchers recognize common stone arrangements and recommend the best move in those situations. Expert rules identify important strategic regions, estimate territorial balance, and suggest candidate moves. Selective search resolves local questions about the life-or-death status of a block or a group of stones.

Researchers throughout the world continue to work on computer Go and gather regularly at computer Go tournaments to pit their updated programs against each other. Currently, the top programs in the world include Handtalk/Goemate, by Zhixing Chen; Go4++, by Michael Reiss; Haruka, by Ryuichi Kawa; Many Faces of Go, by David Fotland; and GNUGo, by multiple developers.

Learning Opening Strategy

In this section, we motivate our approach to learning opening strategy, provide details of our experimental methodology, and describe some features of Go positions that can be supplied as input to a Go evaluation function.

Temporal Difference Learning and Go

Like many other strategy games, the moves played in the early part of a Go game dictate game play for the rest of the game and have the greatest influence on the outcome. At the beginning of a game, players place stones to stake loose territorial claims and to shape the flow of attack and defense in the middle game. Localized battles involving the life-or-death status of a group of stones tend to occur later in the game. Hence, opening strategy focuses more on developing desirable stone patterns than on finding tactical maneuvers to capture a group of stones.

We attempted to learn an evaluation function for opening strategy in Go using a neural network trained with temporal difference learning. We chose a neural network representation because it seems well-suited for the emphasis on pattern recognition in opening strategy, and we chose temporal difference learning, a kind of reinforcement learning, because we wanted the system to learn by self-play and by play against others.

Unlike supervised learning, reinforcement learning relies only on periodic feedback or reinforcement and makes it possible to learn when classified training data sets are unavailable. Temporal difference (TD) learning methods work by using successive predictions of an environment as it evolves over time (Sutton 1988). For Go, the inputs might be descriptions of board positions from move to move over the course of a game, and the outputs might be scalar values representing the probability of a win or the relative desirability of a position.

A traditional Backpropagation approach to neural network learning adjusts the network weights based on the error between the output generated from a particular input and the correct output (Hertz, et.al. 1991). Since the correct output is unavailable to a TD-learner, it adjusts the weights based on the difference between the current network output and the output some time in the future (e.g., one move later). It does not matter if this output is somewhat inaccurate as long as the learner periodically receives error-free output. Such output comes at the end of each game when the outcome is no longer in doubt.

Experimental Methodology

In designing our experiments, we focused on two related tasks. First, whereas others have previously investigated the effectiveness of neural networks and TD-learning for Go in general (Schraudolph, et.al. 1994), we wanted to determine their effectiveness for learning opening strategy in particular. Second, since the true value function over raw board positions is likely very non-linear, we wanted to determine whether and how much the inclusion of various higher-level features of the position would improve the learned evaluation function.

For computational reasons, we limited all our experiments to 9×9 boards. We created 16 computer players, each of which used an evaluation function consisting of a fully connected two-layer feed-forward neural network with 40 hidden nodes and one output node representing the relative score differential between the black and white players. The players differed only by the type and number of inputs used. Of the 5 features described below, each player took as input the raw board position (Feature A) and a selection from four additional features (Features B, C, D, and E). In the remainder of this paper, we refer to a player by the features it takes as input, e.g., Player ACD takes Features A, C, and D.

Typically, the learning phase would involve playing a program against itself or another opponent for a series of complete games and using each game's outcome to adjust the neural network weights. Because we wanted to determine which player stands better at the end of the opening, we took a new approach. For each player, we played out an average length opening (10 moves per side on a 9×9 board) and then handed off the position to GNUGo, which we used as a Go "expert". GNUGo finished the game by playing stones for both sides, and the result was treated as the correct output for the final opening position. Over the long run, the side with a better position after the opening should end up winning the game. This approach focused our learning and testing on the opening moves, and it had the added benefit of reducing the interval between successive reinforcement signals.

Higher-Level Features

Before moving on to the experimental results, we describe the features that were provided as input to the various players. Feature A corresponds to the raw board position

and is supplied to each of the 16 players. Features B, C, D, and E are higher-level features that many instructional texts cite as important for choosing a move. While these features could be derived from the raw board position, we hoped that the value function over these features might be smoother than that over just the raw board position and that including them as input might improve the quality of the evaluation function or the speed of learning.

Feature A indicates the status of an intersection: black stone, white stone, or unoccupied. It also indicates whether the intersection is playable on the next turn.

Feature B indicates the distance from the intersection to the two closest edges of the board. This information may be useful because board edges and corners greatly influence game play.

Feature C indicates the number of liberties for the block to which the stone at that intersection belongs. In general, a block with many liberties is stronger and less likely to be captured.

Feature D indicates the total number of stones in the block to which the stone belongs. This information may be useful because larger groups tend to be difficult to kill.

Feature E indicates the number of friendly and enemy stones that are close to the block to which a stone belongs. Nearby friendly stones can help a block to attack and defend. Likewise, nearby enemy stones can hurt its ability to do so.

Results and Analysis

We trained each of the 16 players by self-play for 2000 games. To encourage exploration and to avoid repeated training over the same sequence of moves, we added a very small random component so that occasionally a move other than the "best" move was played.

To test the effectiveness of our learning, we played 100 games between Player A and Player Random, a player that randomly chooses a move among all legal moves with uniform probability. As in the learning phase, each player made 10 moves, and then GNUGo played out the game to determine the winner. Player A won 71 of the 100 games. This result is encouraging, though with improved learning, we expect Player A to win even more games.

To evaluate the benefit of including higher-level features in the input to the neural network, we played 100 games each between all 16 players in round-robin fashion. Table 1 shows the percentage of games won by each player against all other players. In general, players that used more features won more games; this suggests that including those features either improved the evaluation function's quality, the learning rate, or both. As expected, Player ABCDE, which used all four additional features, performed relatively well against the other players. Somewhat surprisingly, Player ADE won marginally more games than Player ABCDE (though in their head-to-head matchup, Player ABCDE won 53 of 100 games). In a few cases, adding an individual feature actually caused *worse* performance. For example, Players AC and AE both won fewer games overall than Player A.

Player	A E	A C	A	A B C	A B D	A B	A C E	A B E	A C D	A D	A B C D	A B C E	A B D E	A C D E	A B C D E	A D E
Win %	42	43	46	46	46	48	48	49	50	52	54	54	54	55	55	56

Table 1: Winning percentages of each player in a round-robin tournament with all other players.

To evaluate the incremental benefit of including each individual high-level feature, we considered all of the players that did not use that feature and computed the percentage improvement in number of games won when that feature was added. Table 2 shows the average number of additional games won when each of the four features was added. Overall, each feature showed a positive improvement when included in the input. The most beneficial feature was D, the size of a stone's block. The next most beneficial feature was E, the number of nearby friendly and enemy stones. We believe that Feature B, the distance from the edges, is more relevant for larger board sizes, where there is much more of a center region. Also, we suspect that Feature C, the number of liberties, becomes a greater factor in the middle game, when life-or-death issues occur more frequently.

Feature	**Average improvement**
B	4%
C	3%
D	13%
E	8%

Table 2: Average number of additional games won when the individual feature was added to the input.

While it is not surprising that the players using more features generally did better than players using fewer, we expected there to be a larger difference in winning percentage between the best and worst players. One possible explanation is that the degree of learning was not high enough for the additional feature information to make a large difference. Another possible explanation is that the features we chose are beneficial but not critical to accurate move evaluation.

Previous Work

Machine learning has long been an appealing approach to constructing evaluation functions for strategy games. It promises not only to reduce the level of human effort required but also to identify and represent knowledge that may not be easy to encode manually. The challenges include choosing an appropriate representation language,

devising an efficient learning algorithm, and obtaining beneficial training experience.

When classified training data sets are available, traditional supervised learning approaches have been used to learn various aspects of game play in Go. Using a database of expert games and treating each played move as correct and every other legal move as incorrect, Dahl trained a neural network to identify stone patterns with "good shape" vs. those with "bad shape" (Dahl 1999). In related work, Enderton used neural networks for move ordering and forward pruning of selective search in his Golem Go program (Enderton 1991), and Stoutamire used hashed sets of patterns rather than neural networks to identify good and bad shape (Stoutamire 1991). Using neural networks in conjunction with automatic feature extraction methods and informed pre-processing, van der Werf and colleagues learned to predict good local moves from game records (van der Werf, et.al. 2002).

Among reinforcement learning approaches, Tesauro first showed how to apply TD-learning of neural networks to strategy games with his TD-Gammon program, a world class backgammon player (Tesauro 1995). Schraudolph and colleagues applied the same approach to Go with more modest success (Schraudolph, et.al. 1994). Enzenberger attempted to integrate this approach with manually encoded expert Go knowledge in his Neurogo program (Enzenberger 1996).

Conclusions and Future Work

In this paper, we described an approach for learning opening strategy in the game of Go by training a neural network evaluation function using TD-learning. To focus on opening strategy, we played only the opening moves and then employed an "expert" Go program to play out and score positions at the end of the opening. We also presented an experimental methodology in which we trained 16 computer players, each taking as input a different combination of four higher-level features, and we compared the resulting players with each other. While there remains ample room for improvement, the experimental results indicate that our approach is effective for learning opening strategy and that all four higher-level features help improve the quality of the learned evaluation function, the learning rate, or both.

While this research itself is specific to the game of Go, it has ramifications for other work in machine learning. First, it provides an example of how TD-learning of neural networks can be applied even in situations where the interval between reinforcement signals is normally quite long. Second, it provides a methodology for comparing the relative benefit of various features to the quality of a learned evaluation function.

Looking forward, we hope to build upon our work in several ways. First, we plan to try other network structures, input features, and output representations, e.g., those that simply represent the game-theoretic value of a position. Second, we plan to run experiments on larger board sizes. While our approach is directly applicable to 13 x 13 and 19 x 19 boards, it will require much more computation time, most of which is used by GNUGo to play out games. Third, we plan to learn other aspects of game play besides opening strategy. One possibility would be to learn *joseki*, the common patterns of play that are often hard-coded into the pattern matchers of many programs. Fourth, we plan to explore ways to combine *a priori* Go knowledge with our learning approach. Like earlier efforts (Enzenberger 1996), we hope to develop effective methods for integrating learning and other AI techniques in a Go-playing system.

Acknowledgements

This work was supported by the National Science Foundation under Grant No. 9876181, and by Middlebury College.

References

Dahl, F. 1999. Honte, a Go-Playing Program Using Neural Nets. *International Conference on Machine Learning Workshop on Machine Learning in Game Playing*. http://www.ai.univie.ac.at/icml-99-ws-games/.

Enderton, H. 1991. The Golem Go Program. Carnegie-Mellon University Technical Report #CMU-CS-92-101.

Enzenberger, M. 1996. The Integration of A Priori Knowledge into a Go Playing Neural Network. Internet. http://www.markus-enzenberger.de/neurogo.html.

Hertz, J., Krogh, A., and Palmer, R. 1991. *Introduction to the Theory of Neural Computation*. Addison-Wesley.

Kim, J. and Soo-hyun, J. 1997. *Learn to Play Go, Volume 1, 2nd Edition.* Good Move Press.

Müller, M. 2000. Computer Go. In *Artificial Intelligence Journal* 134(1-2): 145-179.

Schraudolf, N., Dayan, and P., Sejnowski, T. 1994. Temporal Difference Learning of Position Evaluation in the Game of Go. In *Advances in Neural Information Processing Systems* 6. Morgan Kaufmann.

Stoutamire, D. 1991. Machine Learning, Game Play, and Go. Case Western Reserve University Report #TR91-128.

Sutton, R. 1988. Learning to Predict by the Methods of Temporal Differences. In *Machine Learning* 3:9-44.

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon. In *Communications of the ACM* 38(3):58-68.

van der Werf, E. Uiterwijk, J., Postmas, E., and van den Herik, J. 2002. Local Move Prediction in Go. In *Proceedings of the 3rd International Conference on Computers and Games.* Edmonton, Canada.