Exploring GnuGo's Evaluation Function with a SVM

Christopher Fellows
Computer Science
Cornell University
ckf5@cornell.edu

Yuri Malitsky
Computer Science
Cornell University
ynm2@cornell.edu

Gregory Wojtaszczyk
Computer Science
Cornell University
grw24@cornell.edu

Abstract

While computers have defeated the best human players in many classic board games, progress in Go remains elusive. The large branching factor in the game makes traditional adversarial search intractable while the complex interaction of stones makes it difficult to assign a reliable evaluation function. This is why most existing programs rely on hand-tuned heuristics and pattern matching techniques. Yet none of these solutions perform better than an amateur player. Our work introduces a composite approach, aiming to integrate the strengths of the proved heuristic algorithms, the AI-based learning techniques, and the knowledge derived from expert games. Specifically, this paper presents an application of the Support Vector Machine (SVM) for training the GnuGo evaluation function.

Introduction

Go is a deterministic, perfect information, zero sum game between two players. The rules are simple, the two players alternate placing black and white stones on a 19x19 grid board trying to surround as much territory (empty intersections) as possible. The only time a stone is removed from the board is when it is completely surrounded by the opposing player's pieces. The game ends when both players pass on successive turns. However, despite the simplicity of the rules, the game is very difficult to master.

Up to this date the best programs have not reached yet the strength of amateur dan players. The large branching factor makes searching ahead infeasible while the complicated net of influences among the stones makes it difficult to create a reliable evaluation function. Yet these complications only make Go an ideal testing ground for AI techniques.

In this paper we experimented with training an evaluation function with supervised learning. As a basis, the publicly available GnuGo (Bump et al. 2005) was used, currently one of the top-ranking Go programs which relies mainly on hand-tuned heuristics and pattern matching techniques. For the learning algorithm, we turned to Thorsten Joachim's SVM-light (Joachims 1999) because of the program's ability to handle large data sets and easy access to controlling parameters.

The results suggest that the SVM could be used for analysis and selection of the optimal evaluation function based on the combination of heuristic and mathematical models.

GnuGo's Evaluation Function

GnuGo (Bump et al. 2005) is a publicly available Go program that is the currently ranked as the fourth best engine. To select the played move, the program follows a multistage approach. First, an inner representation of the board, consisting of the chains of connected stones, is compiled. These chains are then analyzed by a variety of pattern matching algorithms called *move generators*. If the generators determine that a move fulfills a certain criteria, like finishing a known combination or attacking an opponent's chain of stones, the move is assigned a corresponding *reason*.

When the generators are finished, each move's reasons are evaluated to produce the characteristic values shown in Table 1. The values are then evaluated and combined using a set of hand-tuned rules, and the move with the highest resulting value is played. Our goal is to determine the scope of positions these hand-tuned rules can cover.

Value	Description
Territorial Value	The total amount of territory the player is expected to gain by playing this move
Strategical Value	Measure of the effect the move has on the safety of all chains of stones on the board
Max Pos Shape	How good is the best shape formed by this move as determined by the pattern matching code
Max Neg Shape	How bad is the worst shape formed by this move as determined by the pattern matching code
Num Pos Shape	Number of good shapes formed by this move as determined by the pattern matching code
Num Neg Shape	Number of bad shapes formed by this move as determined by the pattern matching code
Follow-up Value	Value which may accrue if the player gets two moves in a row in local area
Influence Follow-up Value	How much territory can be gained if the player gets two consecutive moves in local area
Reverse Follow-up Value	The value the opposing player gains if allowed two consecutive moves in a local area
Secondary Value	Given for move reasons which by themselves are not sufficient to play the move
Min Value	The minimum value this move can get as determined by the pattern matching code
Max Value	The maximum value this move can get as determined by the pattern matching code

Table 1: Description of GnuGo's characteristic values

SVM Training

Training is done on a collection of board positions evaluated by GnuGo to produce a set of the characteristic values of the non-zero valued moves. First, these values are normalized for each game by subtracting the values of the correct move from all others. The SVM optimization technique then finds the hyperplane that separates the correct moves from all others.

Each game can potentially have up to 381 data points associated with it, resulting in exceedingly large data sets for only a handful of games. SVM-light (Joachims 1999) is a program specifically designed for handling learning tasks with many training examples. The overall optimization problem solved by the program is presented below:

$$\begin{array}{ll} \textbf{minimize:} & P(\vec{w}, b, \vec{\xi}) = \frac{1}{2}\vec{w}\cdot\vec{w} + C\sum_{i=1}^{n}\xi_i \\ \textbf{subject to:} & \forall_{i=1}^{n} : y_i[\vec{w}\cdot\vec{x}_i + b] \geq 1 - \xi_i \\ & \forall_{i=1}^{n} : \xi_i > 0 \end{array}$$

In our case, the **x** is the vector of characteristic values for each move. A positive y denotes the correct move, while all other moves are classified as negative. To make sure that we find the correct separation, we use a hard margin and so set C to be large (one thousand). The resulting **w** vector is then used as the weights.

Results

For generating our training set, we turn to the Computer Go Test Collection (Mueller 1995) which is maintained by the Computer Go Group at the University of Alberta. The advantage of this collection is that it is subdivided into ten problem types that can occur in a game, including endgame, life-death, escape capture, low liberties, etc.

Of the 542 problems present, GnuGo currently solves 380. Training however, is done only on the 187 correctly solved problems requiring the actual correct move instead of avoiding the provably bad one. For the resulting 7,000 data points, the SVM finds a perfect separation (Table 2). To show that these weights were not over-fitted, they were tested on 2,000 positions compiled from a collection of professional title matches. Of these, GnuGo solves 316, while the trained weights solve 314 of these and nothing else. This result shows that the learned weights perform close to GnuGo's evaluation function.

To explore the scope of the characteristic values, unsolved problems were added to the training set. In all cases, the SVM was able to improve, but to a varying degree for different puzzle types. Table 2 shows results from two problem type categories: cut connect and escape capture. The first column presents weights from the previous training based on GnuGo solved games, while the second is a new extended set resolved with the SVM approach. The thing to note is the magnitude of the change of some of the weights. It is likely that the static values are the ones that are constraining the learning and must be modified.

	GnuGo	**SVM**	**Cut / Connect**		**Escape Capture**	
Number of Games	187	187	50	59	15	21
Territorial Value	1	14.47	15.1	20.0	0.43	3.97
Strategical Value	1	14.74	14.2	20.3	0.02	-2.8
Max Pos Shape	*	9.43	4.05	44.5	0.75	-2.5
Max Neg Shape	*	-8.93	-8.3	3.40	-0.3	0
Num Pos Shape	*	3.75	2.58	-33	0.42	21.8
Num Neg Shape	*	-4.46	-4.2	-60	-0.2	0
Follow-up Value	$	1.85	0	5.62	0	2.03
Influence Follow-up	$	7.35	4.79	6.54	-0.3	2.16
Reverse Follow-up	$	9.46	0	0	0.01	5.03
Secondary Value	0.05	0	0	0	0	40.3
Num Connected	*	2.74	1.67	10.1	0.02	8.02
Min Value	#	5.22	0.22	21.3	0.01	-0.1
Max Value	#	0.03	0.01	0.10	0	0

Table 2: Comparison of GnuGo's values and those calculated by the SVM. The Cut/Connect and Escape Capture show the weights after training on solved problems and after unsolved problems were added.

** values used in a non-linear combination (e.g. 1.05^x)*

$ min-max combination (e.g. min(min(x+y+z , x+2y) , t+x))

# values used in if-else statements

Future Work

The project establishes a framework connecting the Gnu Go engine, a SVM training approach, and a collection of expert games. Using a linear kernel it successfully approximates the hand-tuned heuristics of Gnu Go leading to an identical level of play. We expect to employ this approach as a platform for developing a more advanced evaluation function based on non-linear kernels and an optimal combination of heuristic and mathematical models.

Acknowledgments

We would like to thank Prof. Joachims for suggesting the investigation of the SVM approach.

References

Bump, D. et al. 2005. GnuGo http://www.gnu.org/software/gnugo/gnugo.html

Joachims, T. 1999. SVM-light http://svmlight.joachims.org/

Mueller, M. 1995. Computer Go Test Collection http://www.cs.ualberta.ca/~games/go/cgtc/