Improving Monte Carlo Go

Goncalo Mendes Ferreira
gonmf@sapo.pt

***Abstract*— The game of Go remains one of the few deterministic perfect information games where computer players still struggle against professional human players. In this work two methods of derivation of artificial neural networks – by genetic evolution of symbiotic populations, and by training of multilayer perceptron networks with backpropagation – are analyzed for the production of a neural network suitable for guiding a Monte Carlo tree search algorithm (MCTS). This last family of algorithms has been the most successful in computer Go software in the last decade. Recently, deep convolutional neural networks have shown much promise in combination with MCTS. These require computational facilities that many computers still don't have however. This work explores the impact and integration of simpler neural networks for the purpose of guiding Monte Carlo tree searches, and improving state-of-the-art MCTS programs.**

Keywords: Go, MCTS, SANE.

I. Introduction

Go is an ancient board game known for its difficulty for computers to play. On the most common board size used, an average game lasts 240 turns and has 2×10^{172} legal game positions [7] – an obstacle to more traditional board game AI approaches.

The world of computer Go was revolutionized between 2002 and 2006 with the invention of Monte Carlo tree search; and again in 2014 to the present day with the application of Deep Convolutional Neural Networks for move prediction and integration with MCTS. In March 2016 for the first time a computer program defeated one of the strongest players in the world.

This work focuses on Monte Carlo tree search, and its implementation on an actual competitive program; as well as the integration of simple networks for play suggestion. The end result was named Matilda.

II. The game of Go

Go is an abstract, deterministic, perfect information, adversarial, two-player, turn-based game. The first player places a black stone on one of the 19x19 intersections of the board. The second player plays with white stones. The game progresses with the players alternating in either placing a stone in an empty intersection, or passing their turn.

A possible start of the game can be seen in Fig. 1. To win the game it is necessary to have the most territory, and to have the most territory it is necessary for the players stones to remain on the board. The stones are removed when their number of liberties reaches zero. The liberties are the unique adjacent empty intersections of a group of stones. Stones **6,8** pictured in Fig. 1 have four liberties while stones **5,7,9** have six. We can say that the white player has a larger influence on the corner of the board, and this intuitive notion can be understood as it being more likely for that area to belong to white, in the end of the game. The match is scored for each player as the number of stones on the board and the area they circumscribe.

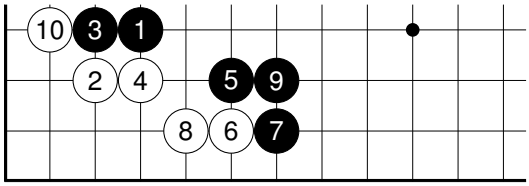

Fig. 1. Play in the corner

III. Monte Carlo Go

In Go the problem state space is too large for minimax state search strategies. We can abstract the large state space to a Markovian decision problem (MDP), and use Monte Carlo tree search to estimate the immediate action that maximizes our chance of winning. We learn the reward distribution – and therefore best transition for the player, by random sampling. Each sample consists in navigating the space until a termination criteria, and registering the outcomes of each first transition of the sample. After a satisfying amount of samples, we select the transition with higher average outcome.

To better traverse the search space, a traversion tree is maintained; essentially equivalent to a exploration-exploitation decision problem at every game state traversed. The algorithm is usually divided in four steps, performed repeatedly over the available time:

1) Selection – from the initial state select and traverse valid transitions until a new state is found.
2) Expansion – expand and evaluate the new state.
3) Playout – randomly play the game until the end from the current state, yielding a playout outcome (win or loss).
4) Propagation – propagate the playout outcome across the transitions that led to the newly expanded state, updating their statistics.

In attempting to reduce the regret of our MCTS sampling we can use both statistical policies and domain knowledge (heuristic) policies. The UCB statistical policy was discovered in 2002 [1] and minimizes the expected reward in generic MDP searches. Matilda first made use of UCB, and then integrated domain heuristics. Today the domain knowledge is

sufficient to replace UCB completely, benefiting fully from the use of AMAF [4]. With AMAF each state transition quality is also updated if played later in the simulation.

The biasing of the exploration phase with domain knowledge in Matilda uses progressive bias – a technique where the statistics of the state are initialized with heuristic contributions – quality and equivalency value in number of simulations.

Further techniques are integrated in MCTS: play values over partial board patterns are precalculated; game states that are similar are reused in the game tree search; plays that are very similar are combined into one; remembering the last effective response play is used to speed up traversion [2]; definitions of play criticality, or urgency; effectivity; handicap adjustment (techniques present in [3]) and more are used – making it a state-of-the-art MCTS implementation.

Domain knowledge in Matilda mostly comes in the form of local tactical evaluation functions – often exhaustive with branching restrictions. Strategic reasoning is also performed around clusters of loosely connected groups of stones.

IV. Neural networks

The use of neural networks (NN) has produced a few strong programs before 2016, such as NeuralGo and Neuron. With the invention of DCNN, NN have greatly increased the strength of existing MCTS programs (Aya, Zen) and led to the strongest program today [6]. In Matilda we modestly started our experiments with 2-layer perceptrons with sparse connections. The purpose was to gauge the impact of the smallest of networks on biasing a MCTS program.

The derivation of the networks was tried both with the SANE method [5] – evolving populations of neurons and networks at the same time and evaluating their fitness by playing whole games – and with classic supervised learning with error backpropagation. Before DCNN it seemed impracticable to train a network to play 19x19 Go with supervised learning. The SANE method aimed to tackle this and in our experiments worked well on board sizes up to 9x9. On larger board sizes, however, the game complexity made it's presence felt, and we couldn't observe continued growth. The fitness calculation was tried both as tournament based and against reference opponents.

In both the SANE and error backpropagation experiments we used simple entry features – number of liberties after play and the contents of the board – codified with three input units per board position. The backpropagation experiments used a local focal field – where each neuron was only connected to neurons up to a certain distance way (in the two dimensional board) of the previous network layer, saving neuron connections. The input units were mapped to the corresponding board positions. Board symmetry was not taken advantage of.

It was noticed early on that the network suffered from over saturation – where each output neuron was so seldom activated that a very small learning rate had to be used – and that local focal fields actually benefited the network accuracy (in contrast with fully connected layers). This comes in part as a surprise; but doesn't change the fact that we needed to find the strength peak between value of information and NN calculation penalty.

After determining the ideal parameters of field distance the MLP, when integrated in the final MCTS program, produced marginally positive results. However these were obtained without the efficient calculation of the network with GPU or dedicated hardware, which means that even unoptimized (reaching 73% of the original simulation speed), it contributes enough to be worth its implementation.

Furthermore the benefit is greater the larger the board, which coincides with where the MCTS approach most suffers. The simple features used are also inexpensive to extract since they are already used for tactical restrictions in MCTS.

The network as a prediction model is very weak; with an accuracy of 2% in 19x19 and 11% in 9x9 over a dataset of Computer Go Server (CGOS) games. Instead, in Matilda it is used another heuristic for progressive bias. We classify all legal plays as good ($t1$), neutral ($t2$) and poor (the rest). In tuning we've found near optimal $t1 = 25\%$ and $t2 = 50\%$. We promote the exploration of the good plays, and hinder the exploration of the poor. Simple as it is, the networks reach classification accuracies of over 45% in 19x19 and 80% in 9x9.

V. Matilda

Today Matilda is available as free software[1] and ranks as 3 dan in 9x9[2], with its strength suffering the larger the board. It has made some appearances in online computer tournaments, with its best performance a 2nd place at the Kiseido Go Server (KGS). It supports all major protocols and standards for computer Go. Parameter optimization was performed with genetic, MCTS and linear optimization algorithms. The software for black box tuning CLOP was also used. Unfortunately it is still a very slow process, requiring full games as optimization samples, with each taking several minutes in 19x19 boards.

For knowledge acquisition we used game records from self-play, the KGS, Kogo's Joseki Dictionary and the CGOS.

References

[1] Peter Auer, Paul Fischer, and Jyrki Kivinen. Finite-time analysis of the multiarmed bandit problem. In *Machine Learning*, 2002.

[2] Hendrik Baier and Peter D. Drake. The Power of Forgetting: Improving the Last-Good-Reply Policy in Monte-Carlo Go, 2010.

[3] Petr Baudiš. *MCTS with Information sharing*. PhD thesis, Faculty of Mathematics and Physics of the Charles University, 2011.

[4] Bruno Bouzy and Bernard Helmstetter. Monte Carlo Go Developments. In *Advances in Computer Games Conference ACG-10*, pages 159–174, 2003.

[5] Norman Richards, David E. Moriarty, and Risto Miikkulainen. Evolving Neural Networks to Play Go. *Applied Intelligence*, 8:85–96, 1997.

[6] David Silver and Aja Huang et al. Mastering the Game of Go with Deep Neural Networks and Tree Search, 2016.

[7] John Tromp. Number of legal Go positions, 2016. `http://tromp.github.io/go/legal.html` [Online; accessed 22-January-2016].

[1] Can be downloaded from `https://github.com/gonmf/matilda`

[2] Estimated from play in the Hiroshi Yamashita CGOS instance – `http://www.yss-aya.com/cgos` – where is ranked 2365 ELO.