Temporal-Difference Search in Computer Go

David Silver
Department of Computer Science
University College London
Gower Street, London WC1E 6BT

Richard Sutton, Martin Müller
Department of Computing Science
University of Alberta
Edmonton, Alberta T6G 2E8

Abstract

Temporal-difference (TD) learning is one of the most successful and broadly applied solutions to the reinforcement learning problem; it has been used to achieve master-level play in chess, checkers and backgammon. *Monte-Carlo tree search* is a recent algorithm for simulation-based search, which has been used to achieve master-level play in Go. We have introduced a new approach to high-performance planning (Silver, Sutton, and Müller 2012). Our method, *TD search*, combines TD learning with simulation-based search. Like Monte-Carlo tree search, value estimates are updated by learning online from *simulated* experience. Like TD learning, it uses *value function approximation* and *bootstrapping* to efficiently generalise between related states. We applied TD search to the game of 9×9 Go, using a million binary features matching simple patterns of stones. Without any explicit search tree, our approach outperformed a vanilla Monte-Carlo tree search with the same number of simulations. When combined with a simple alpha-beta search, our program also outperformed all traditional (pre-Monte-Carlo) search and machine learning programs on the 9×9 Computer Go Server.

Introduction

In this article we demonstrate that an agent can achieve high-performance planning by applying *reinforcement learning* to simulated experience. We focus on the game of Go as a concrete example of a large, challenging environment in which traditional approaches to planning have failed to produce significant success. Recently, *Monte-Carlo tree search* (MCTS) algorithms, have revolutionised computer Go and achieved human master-level play for the first time (Coulom 2006; Gelly and Silver 2007). The key idea of these algorithms is to use the mean outcome of random simulations to evaluate positions. Many thousands of games are simulated, starting from the current position, and *rolling out* games by randomised self-play. A search tree maintains the value of each visited position, estimated by the mean outcome of all simulations that visit that position. The search tree is used to guide simulations along promising paths, by selecting the child node with the highest value or potential value (Kocsis and Szepesvari 2006).

However, MCTS suffers from two major deficiencies: each position is evaluated independently, without any generalisation between similar positions; and the mean value estimates can have high variance. As a result, MCTS can require prohibitively large numbers of simulations when applied to very large search spaces, such as 19×19 Go. In practice, the strongest current Go programs deal with these issues by using domain specific solutions (Gelly and Silver 2007; Gelly et al. 2006). In this article we develop a much more general framework for simulation-based planning that addresses these two weaknesses. Our approach includes Monte-Carlo tree search as a special case, but it uses *value function approximation* to generalise between related positions; and it uses *bootstrapping* to reduce the variance of the value estimates.

In classic games such as chess (Veness et al. 2009), checkers (Schaeffer, Hlynka, and Jussila 2001), and backgammon (Tesauro 1994), *temporal-difference learning* (TD learning) has been used to achieve human master-level play. In each case, a value function was trained offline from games of self-play; this value function was then used to evaluate leaf positions in a high-performance alpha-beta search. However, in challenging environments such as Go, it is hard to construct an accurate global value function (Müller 2002). Instead, we approximate the value of positions that occur in the subgame starting from *now* until termination. This new idea is implemented by re-training the value function *online* in real-time, by TD learning from games of self-play that start from the current position. The value function evolves dynamically throughout the course of the game, specialising more and more to the particular tactics and strategies that are relevant to *this* game and *this* position. We demonstrate that this method, which we call *temporal-difference search* (TD search), can provide a dramatic improvement to the quality of position evaluation.

Temporal-Difference Learning in Go

We learn a position evaluation function for the game of Go, without requiring any domain knowledge beyond the grid structure of the board. We use a simple and efficient representation, based on local 1×1 to 3×3 patterns of stones, to capture intuitive shape knowledge. We evaluate positions using a linear combination of these pattern features, and learn weights by TD learning and self-play.

We use a reward function of $r = 1$ if Black wins and $r = 0$ if White wins, with no intermediate rewards. The

Learning	Search	Elo rating
TD learning	None	1050
TD learning	Alpha-Beta	1350
TD learning	TD search	2030
TD learning	TD search + Alpha-Beta	2130

Table 1: The Elo ratings established by *RLGO 2.4* on the Computer Go Server in October 2007.

value function $V^\pi(s)$ is the expected total reward from state s when following policy π for both Black and White. This value function is Black's *winning probability* from state s. Black seeks to maximise his winning probability, while White seeks to minimise it. We approximate the value function by a logistic-linear combination of local shape features $\phi(s)$ with weights θ, using two forms of weight sharing to capture rotational, reflectional and translational symmetries,

$$V(s) = \sigma\left(\phi(s) \cdot \theta\right) \tag{1}$$

where σ is the logistic function, $\sigma(x) = \frac{1}{1+e^{-x}}$. Weights are updated by TD learning (Sutton and Barto 1998), by updating Black's winning probability towards Black's subsequent estimate of winning probability at her next move (and vice versa for White),

$$\Delta\theta = \alpha(V(s_{t+2}) - V(s_t))\phi(s_t) \tag{2}$$

Updating values from successor values is known as *bootstrapping*, and is known to significantly reduce variance (Sutton and Barto 1998). Actions are selected by maximising/minimising $V(succ(s,a))$ for Black/White respectively, while selecting random move with probability ϵ.

Temporal-Difference Search in Go

Rather than planning for every possible eventuality, TD search focuses on the subproblem that arises from the current state s_t: how to perform well *now*. In a nutshell, TD search applies TD learning to simulations drawn from the local subproblem. Specifically, each simulation begins from the root state s_t, and uses the current value function $V(s)$ to generate moves for both Black and White (again selecting a random move with probability ϵ), until the simulated game is completed. After each simulated move, the value function is updated by TD learning (Eqns 1 and 2). The key difference is that the weights are specialised to simulations starting from s_t, rather than full games from the start position s_1. Finally, an actual move is selected by maximising/minimising $V(succ(s_t,a))$ and the root state is updated to the new position $s_{t+1} = succ(s_t,a)$. Another search is then started from s_{t+1}, reusing the previous weights as an initial estimate.

Dyna-2: Long and Short-Term Memories

We now develop a unified architecture, *Dyna-2*, that combines both TD learning and TD search. The agent maintains two distinct value functions: using a *long-term memory* $\langle\phi,\theta\rangle$ and a *short-term memory* $\langle\overline{\phi},\overline{\theta}\rangle$. TD learning is applied to train the long-term memory, to learn general knowledge about the game. TD search is applied to train the short-term memory, to learn *local* knowledge about the problem, i.e. corrections, adjustments and special-cases to the long-term memory that provide a more accurate local

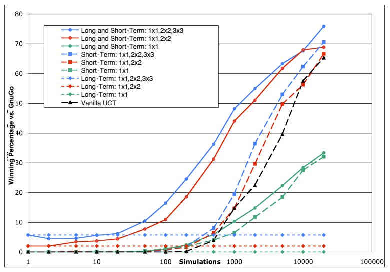

Figure 1: Winning rate of *RLGO 2.4* against *GnuGo* in 9×9 Go, when varying number of simulations. Local shape features from 1×1 up to 3×3 were used in long-term memory (dotted), short-term memory (dashed), or both (solid). Local shape features varied in size. Each point represents winning rate over 1,000 games.

approximation to the true value function. One simple implementation of this idea uses the same features $\phi = \overline{\phi}$ for long and short-term memories; in this special case the short-term weights can simply be re-initialised to long-term weights, $\overline{\theta} \leftarrow \theta$, at the start of each new game; TD search then proceeds as described in the previous section. Figure 1 shows the performance of our Go program, *RLGO 2.4*, using either TD learning with a long-term memory, TD search with a short-term memory, or the combination of both.

Combining TD Search and Alpha-Beta Search

TD search can be combined with traditional search methods such as alpha-beta search. The value function $V(s)$ is used as a leaf evaluation function. As the game progresses, the evaluation function adapts online to specialise to the current position. Table 1 gives the Elo ratings of our Go program, *RLGO 2.4*, using various combinations of TD learning, TD search, and alpha-beta search.

References

Coulom, R. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *5th International Conference on Computer and Games*, 72–83.

Gelly, S., and Silver, D. 2007. Combining online and offline learning in UCT. In *17th International Conference on Machine Learning*, 273–280.

Gelly, S.; Wang, Y.; Munos, R.; and Teytaud, O. 2006. Modification of UCT with patterns in Monte-Carlo Go. Technical Report 6062, INRIA.

Kocsis, L., and Szepesvari, C. 2006. Bandit based Monte-Carlo planning. In *15th European Conference on Machine Learning*, 282–293.

Müller, M. 2002. Computer Go. *Artificial Intelligence* 134:145–179.

Schaeffer, J.; Hlynka, M.; and Jussila, V. 2001. Temporal difference learning applied to a high-performance game-playing program. In *17th International Joint Conference on Artificial Intelligence*, 529–534.

Silver, D.; Sutton, R. S.; and Müller, M. 2012. Temporal-difference search in computer go. *Machine Learning* 87(2):183–219.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: an Introduction*. MIT Press.

Tesauro, G. 1994. TD-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation* 6:215–219.

Veness, J.; Silver, D.; Blair, A.; and Uther, W. 2009. Bootstrapping from game tree search. In *Advances in Neural Information Processing Systems 19*.