01 Jan 1997

Adaptive Critic Design in Learning to Play Game of Go

R. Zaman

Danil V. Prokhorov

Donald C. Wunsch
Missouri University of Science and Technology, dwunsch@mst.edu

Recommended Citation

R. Zaman et al., "Adaptive Critic Design in Learning to Play Game of Go," *Proceedings of the International Conference on Neural Networks,1997*, Institute of Electrical and Electronics Engineers (IEEE), Jan 1997. The definitive version is available at https://doi.org/10.1109/ICNN.1997.611623

Adaptive Critic Design in Learning to Play Game of Go

Raonak Zaman
Raonak@eesun1.ee.ttu.edu

Danil Prokhorov
Prokhor@eesun1.ee.ttu.edu

Donald C. Wunsch II
Dwunsch@coe2.coe.ttu.edu

Applied Computational Intelligence Laboratory
Electrical Engineering
Texas Tech University
Box. 43102, Lubbock, TX 79409-3102
http://www.acil.ttu.edu

Abstract

This paper examines the performance of an HDP-type adaptive critic design (ACD) of the game Go. The game Go is an ideal problem domain for exploring machine learning; it has simple rules but requires complex strategies to play well. All current commercial Go programs are knowledge based implementations; they utilize input feature and pattern matching along with minimax type search techniques. But the extremely high branching factor puts a limit on their capabilities, and they are very weak compared to the relative strengths of other game program like chess. In this paper, the Go-playing ACD consists of a critic network and an action network. The HDP type critic network learns to predict cumulative utility function of the current board position from training games, and, the action network chooses a next move which maximizes critics' next step cost-to-go. After about 6000 different training games against a public domain program, WALLY [1], the network (playing WHITE) began to win in some of the games and showed slow but steady improvements on test games.

Introduction

Go is a deterministic, perfect information, zero-sum game of strategy between two players. Players take turns putting black and white pieces (called stones) on the intersection of the lines in a 19x19 board (usually 9x9 for computer programs, including ours). Once played, a stone cannot be moved, unless captured by the other player. A player can pass any time. The object of the game is to surround territory and/or opponent's stones. Adjacent stones of the same color forms groups; an empty intersection adjacent to a group is called its liberty. A group is captured, when its last liberty is occupied by the opponent. To prevent loops, it is illegal to make moves that recreates prior board position (rule of Ko). The game ends when both players pass in successive turns [2].

Most computer game playing algorithms use minimax techniques along the move tree for several ply (one ply consists of two consecutive moves, one by each player) along with static position evaluator to pick the best move. There are several reasons why this approach is not efficient for Go. First, for normal board sizes the number of legal moves at each position or branching factor is much higher than in chess. Second, many situations in Go require very deep reading in order to assess correctly (since the stones do not move around, a human player can look ahead more reliably than in chess, in some cases for 60 ply). Third, there is no simple evaluation function that could be applied to the leaf positions in the minimax move tree.

In his Neurogammon project, Tesauro (1993) used Temporal difference, TD(λ), algorithms for the prediction evaluation function at different board positions [3]. TD(λ) methods are incremental learning procedures specialized for prediction problems where the sensory inputs are applied in sequence (Sutton, 1988) [4]. TD(λ) algorithms adjusts the weights in a multi-layer network; the incremental weight is given as:

$$\Delta w_t = \alpha\left(P_{t+1} - P_t\right)\sum_{k=1}^{t}\lambda^{t-k}\Delta_w P_k\,, \qquad (1)$$

and, it minimizes the following criterion function:

$$J(w) = \sum_{p=1}^{P}\sum_{k=1}^{N_p}\lambda^{N_p-k}\left(zN_p - G(x_p(k))\right)^2 \qquad (2)$$

In equations (1) and (2), P is the number of examples, e.g., the number of games; N_p is the number of steps in the p^{th} example, which is not known until the outcome is determined; zN_p is the actual outcome (determined by the rule of the game) of the p^{th} example at the end of the game p. Game p consists of states $x_p(k)$, k = 1, 2, .. N_p. $G(x_p(k))$

0-7803-4122-8/97 $10.00©1997 IEEE

is the output of the network when presented with $x_p(k)$; and, $\lambda[0,1]$ is a parameter which is used to place more emphasis on predictions temporally close to the outcome.

Schraudolph et. al (1994) used TD(0) for training the critic in their Go network [5]. Chan et. Al. (1996) showed that non-zero λ gives better learning and the larger the λ is, the better is the performance [6]. In our Go project, we used Heuristic Dynamic programming (HDP) type adaptive critic design [7] for evaluating a Go board. The main differences of HDP with above mentioned three TD(λ) approaches are, it uses an additional utility function (per step cost/reward) in training signal. We used on-line learning since instead of using a fixed set of training games, we trained our network by playing against a public domain AI-type Go-program, WALLY.

An HDP-type critic estimates the function J (cost-to-go) in the Bellman equation of dynamic programming expressed as:

$$J(t) = \sum_{k=0}^{\infty} \gamma^k U(t+k), \tag{3}$$

where, γ is a discount factor for finite horizon problems ($0<\gamma<1$), and U(.) is a non-negative utility function or local cost/reward. The critic is trained forward in time, and it tries to minimize the following error measure over time

$$\|E_i\| = \sum_t E^2(t) \tag{4}$$

and,

$$E(t) = J(t) - \left[\gamma\, J(t+1) + U(t)\right] \tag{5}$$

where, the terms inside the square bracket make the desired signal for time step t, if t is not the terminal state. At the end of the game, the desired signal is simply U(t) or modified U(t) to reflect the outcome of the experiment [7,8]. J(t) is a function of R(t), i.e., the observable states of plant. For our Go-playing ACD, R(t) can be the board representation at step t. The winner in a Go-game is determined by counting areas of two players and the player with more area wins. For the function U(t) we used an incremental area measure from board R(t-1) to R(t). When area associated with R(t-1) is larger than that of R(t) (loss of area between two steps, (t-1) and t), U(t) is set to zero since U(.) is strictly non-negative by the principle of dynamic programming. This means that, the critic will only learn to predict outcome for one player. In our experiments, we trained the critic to play as WHITE, and, WALLY played as BLACK.

Network architecture

WALLY, a weak public domain program (rating ~30 Kyu), is used to provide the BLACK moves. So, only half of the states were created by the ACD. But, the critic actually can see all the states because utility after WALLY move will be zero. The action network is an algorithm that picks a move from all legal moves which will maximize J for next board configuration. So, J will seek to maximize WHITE's area. By legal moves, we mean all empty board positions, except those violating the rule of Ko or involuntary suicide rule. These rules are worth giving to the network rather than starting from zero knowledge to give the network a structure. After training 6000 games using gradient descent weight update an 81x21x1 network learned to defeat WALLY in some games. All the hidden nodes have bipolar sigmoid activation and the output node has sigmoid activation. The slope of activation functions are 0.5 in all cases. The block diagram of the training process is shown in Fig. 1.

The critic sees each states after both WALLY and the action network moves. The action network picks only WHITE moves, so it sees only even-numbered states. The action network sees the state R(t) and picks up a WHITE move that will maximize J(R(t+1)). During first 6000 training games, the initial state was always the blank board, i.e., we are training on even play.

The U(.) function is given by:

$$\begin{aligned} U(t) &= \text{util}(t) - \text{util}(t-1)\ ;\ \text{if util}(t) > \text{util}(t-1) \\ &= 0, \text{otherwise.} \end{aligned} \tag{6}$$

where,

$$util(t) = \eta \frac{N_W}{N_W + N_B} + \nu \frac{A_w}{A_w + A_B} + \rho \frac{P_B}{P_W + P_B} \tag{7}$$

To avoid singularity, any of the terms on right side of Eq. (7) may be reduced to zero if its denominator is zero. N_w and N_B are the number of WHITE and BLACK stones on board, respectively. Aw and A_B are the area occupied by WHITE and BLACK stones, respectively. P_B and P_W are BLACK and WHITE prisoners held by the opponent, respectively . One's utility measure increases with increasing his own occupied area or capturing more of opponent's stone. The first term is included to teach critic not to make unnecessary PASS moves. The third term in the utility will train the critic to weigh the importance of attacking play. In our experiment, we did not explicitly include the first term but our area measurement actually gave (A_w + N_w) and (A_B + N_B) instead of simply A_w and A_B.

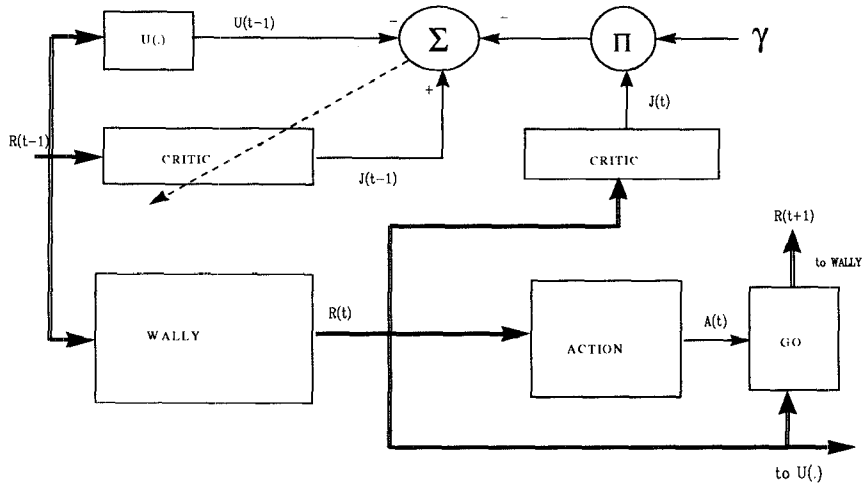

Figure 1. Block diagram of the Go-playing network.

We approximately chose the weights ν=0.8 and ρ=0.2.The desired signal for the terminal state (end of one game) is given by:

$$D(T) = \frac{A_W}{A_W + A_B}. \qquad (8)$$

Training Strategies and Results

During the first 4000 training games, the action network selected moves by Gibb's sampling, as in Chan et. al. (1996), to make variations in play since WALLY tends to repeat the same plays. This is necessary only at the beginning because during that period the final outcome is very close to zero in most cases. Thus, the training signals are large, and action network will quickly fall into some local minima to adopt one line of play only. Gibb's sampling will provide a tool to explore multiple paths in the depth-first search tree. At the end of 3000 training games, the network was occupying an average of 29 squares among a total of 81 squares. During this period all initial states were the empty board. Beyond the 4000th game the action network started to pick moves that maximized the critic's evaluation of the next board. The network started to build up more area (on a moving average of last 200 games) with training and registered its first win shortly after 5500th training game. The critic's weight was saved (GO.WT) after 6000 training games when the network was winning in 1% games and occupying more than 34 squares (close defeat) in 20% of the games. Gamma was 0.95 during this training process.

We performed four different training and testing cycles using this saved weight (GO.WT) as the initial weight. Gamma were 0.0, 0.7, 0.9, 0.95 in those experiments respectively. We forced WALLY to play different games each time by selecting first two moves (one BLACK and one WHITE) at random. 100 test games (no online learning) were interleaved between each 500 training games (online learning, not on fixed training set).The number of wins and close losses by the network for a total of 4600 training and testing games in four different cases are shown in TABLE I.

Table I

γ	No. of Wins	No. of losses by less than 6 points
0.0	14	27
0.7	27	37
0.9	41	194
0.95	44	69

As a rule in the game Go, at the end of the game Black's score is reduced by some points to nullify the advantage of placing the first stone. This particular point called "Komi" is generally 5.5. With Komi in place, all the losses with less than 6 points will be counted as White wins.

Conclusion

We have compared the performance of an HDP type adaptive critic design, that can learn to play Go from zero knowledge, for different values of discount factor. The result clearly indicates regular improvement in total winning number for larger value of discount factor, gamma. This simple ACD design of the game Go does not represent a strong Go playing network yet, but it does demonstrate that, the principle of dynamic programming can be utilized to incorporate machine learning in the game of Go. Go is a game of complex strategies at each stages of the game; success of ACD depends solely on the quality of the J function. To improve the relative accuracy of J function from board to board, we are currently experimenting with a simultaneous recurrent network [9] to predict J. Also, the utility function with more Go-related knowledge (e.g., liberty of a group or stone, no. of 2-eye groups, control over the neighboring empty space, etc.) will aid in forming a meaningful J function.

Acknowledgment

We deeply acknowledge support from Texas Tech Center for Applied Automation and Research and the National Science Foundation Neuroengineering Program Grant No. ECS-9413120.

References

1. Bill Newman, "Wally - a Simple Minded Go-program," Shareware Go program available by anonymous ftp from ftp://imageek.york.cuny.edu/nngs/Go/comp/.
2. "The game of Go," by Arthur Smith, Charles Tuttle Co., Tokyo, Japan, 1956.
3. S. Sutton, " Learning to predict by the method of temporal differences," Machine learning, No. 3, 1988, pp. 9-44.
4. G. Tesauro, "Practical Issues in temporal difference learning," Machine learning, No. 8, 1992, pp. 257-278.
5. N. N. Schraudolph, P. Dyan, T. J. Sejnowski, "Temporal learning of position evaluation in the game of Go," Advances in Neural Information Processing, Vol. 6, 1994, pp. 817-824.
6. H. W. Chan, I. King, J. C. Lui, "Performance analysis of a new updating rule for TD(λ) learning in feedforward networks for position evaluation in Go game," in Proceedings of the ICNN, Vol. 3, Washington DC, 1996, pp. 1716-1720.
7. D. Prokhorov, D. C. Wunsch II, "Adaptive critic designs," accepted in IEEE Trans. On Neural Networks.
8. D. Prokhorov, R. Santiago, D. C. Wunsch II, "Adaptive critic designs: a case study for neurocontrol," Neural Networks, vol. 8, no. 9, 1995, pp. 1367-1372.
9. P. Werbos and X. Pang, "Generalized Maze Navigation: SRN Critics Solve What Feedforward or Hebbian Nets Cannot", in the Proceedings of the Ninth Yale Workshop on Adaptive and Learning Systems, New Haven CT, 1996, pp. 51-58.