

TD Methods Applied to Mixture of Experts for Learning 9x9 Go Evaluation Function

Raonak Zaman Donald C. Wunsch II
Mraonaku@ttu.edu Dwunsch@coe2.coe.ttu.edu

Applied Computational Intelligence Laboratory
Electrical Engineering, Texas Tech University, Lubbock, TX 79409
<http://www.acil.ttu.edu>

Abstract

The temporal difference (TD) method is applied on a committee of neural network experts to learn the board evaluation function for the Oriental board game Go. The game has simple rules but requires complex strategies to play well, and, the conventional tree search algorithm for computer games make poor Go program. Thus, the game Go is an ideal problem domain for exploring machine learning algorithms. Here, the neural networks learned a board evaluation function for Go played on 9x9 board sizes. Two learning algorithms, e.g., hybrid mixture of experts (HME) and Meta-Pi, are used to train the neural network experts. Both algorithms learned good Go evaluation functions and the neural network based Go engines were able to defeat a public domain rule-based program more than 50% of the times. The performances of the mixture networks are compared with that of a single feedforwrd network trained similarly.

Introduction

Go is a deterministic, perfect information, zero-sum game of strategy between two players. Players take turns placing black and white pieces (called stones) on the intersections of the lines in a 19x19 grid, called the Go board. Once played, a stone cannot be moved, unless captured by the other player. The object of the game is to surround territory (empty grids) by one's own stone. Adjacent stones of the same color form strings; an empty intersection adjacent to a string is called its liberty. A group is captured when the opponent occupies its last liberty. To prevent loops, it is illegal to make moves that recreate prior board positions (rule of Ko). A player can pass any time. The game ends when both players pass in successive turns. There are excellent books available on the game of Go [1].

There are several reasons why the conventional tree search algorithm of computer game programming approach is not efficient for Go. First, the number of legal moves at each position (i.e., the branching factor of the search tree) is very large. Second, many situations in Go require very deep reading in order to assess correctly. Third, the formulation of Go evaluation function is very difficult and heuristics are generally used.

The temporal difference learning method (TD) has been used to train neural networks for learning game evaluation functions [2], [3], [4], [5]. The TD method is a reinforcement learning approach and it uses the true final outcome of a multi-step process to update its predictions on the intermediate steps in the process. The TD algorithm thus solves the credit assignment problem (to somehow divide the credit for wins or punishment for losses among the moves made) automatically and can learn a board evaluation function simply by playing the game.

For each pair of successive game positions, the TD method uses the difference between the evaluations of the two positions to modify the earlier position's evaluation [6]. For the prediction problem $P(x, w)$, where experience comes in observation-outcome sequences of the form $x_1, x_2, x_3, \dots, x_m, z$. The TD approach represents the error $(z - P_t)$ as a sum of changes in predictions as:

$$z - P_t = \sum_{k=t}^m (P_{k+1} - P_k), \quad (1)$$

where $P_{m+1} = z$. The value of w is updated by first computing a set of Δw_t 's using:

$$\Delta w_t = \alpha (P_{t+1} - P_t) \sum_{k=1}^t \nabla_w P_k. \quad (2)$$

Each Δw_t depends only on a pair of successive predictions and on the sum of all the past $\nabla_w P_t$ values. The sum of weight change after a complete sequence is:

$$\Delta w = \sum_{t=1}^m \alpha (z - P_t) \nabla_w P_t = \sum_{t=1}^m \alpha (P_{t+1} - P_t) \sum_{k=1}^t \nabla_w P_k. \quad (3)$$

Eq. 3 adds all the past gradients at their full intensity. In a variant of this, called TD(λ), the predictions of observation vectors occurring k steps in the past are weighed according to λ^k for $0 \leq \lambda \leq 1$:

$$\Delta w_t = \alpha (P_{t+1} - P_t) \sum_{k=1}^t \lambda^{t-k} \nabla_w P_k. \quad (4)$$

The main advantage of the TD method is that it can be applied in model free systems; the environment serves as its own model.

TD method was very successful in building the evaluation function for the Backgammon game. Schraudolph et al. [4] and Enzenberger [5] have used the TD(0) method to train neural networks for playing 9x9 Go and obtained encouraging results. The authors of this paper have used heuristic dynamic programming or HDP-type adaptive critic module to learn a Go evaluation function [3]. This paper reports the results of new ideas implemented utilizing the experiences of [3]. Here, we examined several implementations of neural network systems to play the game of Go on 9x9 boards. In our experiments, the importance of input features, self-playing strategy and multiple critics, respectively, have been established and the results are presented in detail for a series of experiments under different conditions.

System Engineering

Network Architecture: The neural networks used the temporal difference learning rule for weight update. The board evaluation neural network is called critic, and the complete system the network. First, single multi-layer feedforward networks are used as the critic. Later, networks made of multiple critics, e.g., mixture of experts are trained and tested. Wally, a public-domain program [7], served as an opponent during the test phase. The network architecture during the training phase shows the connections of the critic(s) with other peripherals to play Go and support learning (Figure 1). It has four modules: critic, action, Go rules and reinforcement.

Critic. The critic is neural network(s) trained by TD learning. Critic estimates an evaluation function for the current board position.

Go rules. This unit checks the legality of moves, returns new board position after a move and scores the game.

Action. It selects a move from all the legal moves at a given board position using the critic's current evaluation function.

Reinforcement. The game outcome (i.e., who won the game) issues the final and reinforcement signal, r , to judge the entire sequence of the current game.

The TD critic gets its input at the game state t , $R(t)$, and calculates the output, $J(w_c, R_t)$. The desired signal for the critic output at step t is given by its output at step $t+1$. The action network picks a move and determines the game state $R(t+1)$. The same critic calculates the desired output for step t , $J(w_c, R_{t+1})$. The weight updates were performed at the end of each game and the weight vector w_c was fixed at w^a during the entire n th game sequence.

The network connections during the test phase are shown in Figure 2. An additional knowledge based unit is added to include the very basic Go knowledge [1]. The critic calculates the J value for rest of the legal moves and picks the move, which results in the board configuration to deliver the highest (lowest) J value. The game continues until there are no legal moves available and then the game is scored.

B. Rich Representation of the Input Board: We compared a raw neural network input representation with another input representation consisting of additional features. In the raw representation, there are 81 input nodes for 9x9 boards and black, white and empty grids are encoded as -1, 1 and 0, respectively. In the richer input format, each board intersection is represented in a tabular format by a 3-input vector to encode the following: (a) if the point is an empty grid: (1) the value of the influence function resulting from neighboring stones [8]; and (b) if the point is occupied by stone: (2) number of stones in a string modulated by its color, and (3) number of string liberties modulated by its color.

Four additional important game features were added to both raw and rich representations: (1) number of black prisoner, (2) number of white prisoner, (3) if white passed the last move, and (4) if black passed the last move.

Thus for 9x9 Go, the network using the rich representation has $(9 \times 9 \times 3 + 4) = 247$ nodes in its input layer. And, the network has only $(9 \times 9 + 4) = 85$ input nodes in a raw board representation.

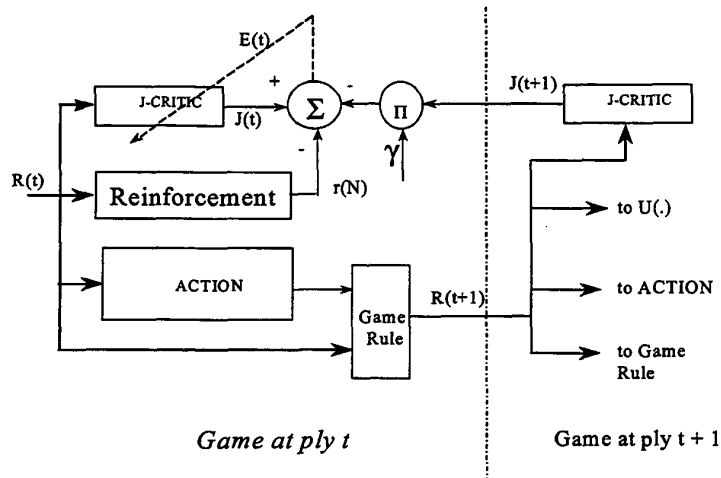


Figure 1: Critic training for the 9x9 Go network. Two consecutive board positions are shown in the game ply t and $t+1/2$. The Action network uses the current critic network to evaluate all board position to select a network move. After the move, the old critic is used to evaluate the next board position and determine the desired signal for the TD learning.

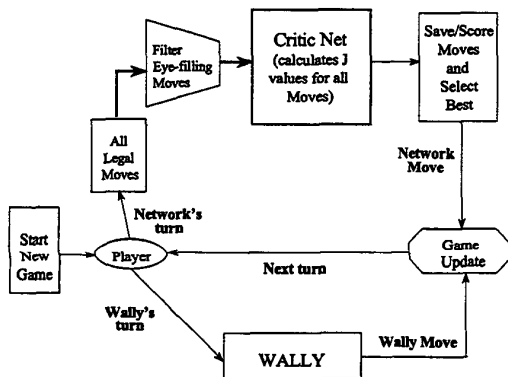


Figure 2: Algorithmic layout during test phase for 9x9 Go. The network picks the move, which results in the board position that gets the best score by critic evaluation.

C. Network Outputs and State Space Exploration:

We used two output neurons, e.g., B and W, of sigmoidal activation to represent the critic's predictions. At the end of game, the desired values of B and W, respectively, for three possible game outcomes are follows: {black_win, white_win, tie} = {(1.0, 0.0), (0.0, 1.0), (0.5, 0.5)}.

A greedy strategy for the move selection would try to maximize the critic's (B - W) value when it is black's turn to move, and minimize it during white's turn, and so on. A greedy strategy during training can thus get the critic stuck into local minima.

To explore a larger state space during training cycles, moves are selected stochastically using Gibb's sampling [4], defined by:

$$P(m_i) = \frac{1}{\sum_i e^{J(m_i)/T}} e^{J(m_i)/T}, \quad (5)$$

$P(m_i)$ is the probability of selecting move m_i , $J(m_i)$ is the critic's board evaluation after move m_i is made. T is called temperature parameter; during training its value is reduced monotonically from 1.0 to 0.2. With a lower temperature, probability of selecting the best move is more than that with a higher temperature.

D. Self Play and Opponent Play:

Two different strategies to select an opponent to play against the network during the training phase have been investigated. In a self-played network, the critic selects moves for both sides using its evolving evaluation function. In the other strategy, the network is trained by playing against an opponent—the critic plays moves for one color and another Go program or a human player plays the other color, then they switches side in the next game, and so on. Our experiments here will show that the reinforcement learning agent achieved better performances in the game Go for 9x9 boards when trained playing against itself rather than against an opponent.

9x9 Go Using Single Feedforward Networks

Three different λ values 0.0, 0.3 and 0.8, respectively, were used for each class of critic training. Raw inputs and rich feature inputs were used in each case and the performances between self-trained and opponent-trained critics were compared. Each critic was trained for 50,000 games and the network played 200 test games after every 1000 training games. Of 200 test games, the network played as black for 100 games and as white for 100 games. Each test game was started from a random initial position with 1 to 7 stones placed on board to ensure varieties in the test games. We define a learning curve as the average margin of victory, i.e., average of (Network_score - Wally_score), in the 200 test games. The learning curves for the various experiments are plotted against the training cycles to track the process.

Thus, the input representations formed (i) raw input critic, and (ii) rich input critic. Again, each type of critic was trained with (i) 2-layer feedforward network with no hidden layer, and (ii) 3-layer feedforward network with 1 hidden layer. We generated four sets of random weights to fit each of the four combinations in the above two settings and trained them by self play and opponent play with the λ value set at 0.0, 0.3 and 0.8, respectively.

Experiments with raw inputs: The critic networks have 85 input nodes and 2 output nodes. Eighty-one input nodes receive the board positions black, white or empty coded as -1/1/0; the other four inputs are global game statistics. All the six critics trained using the raw input features, the 2-layer and the 3-layer MLPs with $\lambda = 0.0$, 0.3 or 0.8, respectively, failed to learn strong Go evaluation function. The best performance among them was achieved by the 2-layer feedforward network trained with $\lambda = 0.3$, which learned to defeat Wally 9.5% of the time at the end of 50,000 training games. The margin of victory improved from an initial value of -40.0 to -24.5 in that case.

Experiments with rich input representation: Here, a 3-input vector represents each board position. The four global game features are also included in the input feature. Thus, the critics have 247 input nodes. The number of output nodes is the same at 2. The learning curves for the critics trained using rich inputs are shown in Figure 3. A positive margin of victory is the desired outcome for the training process.

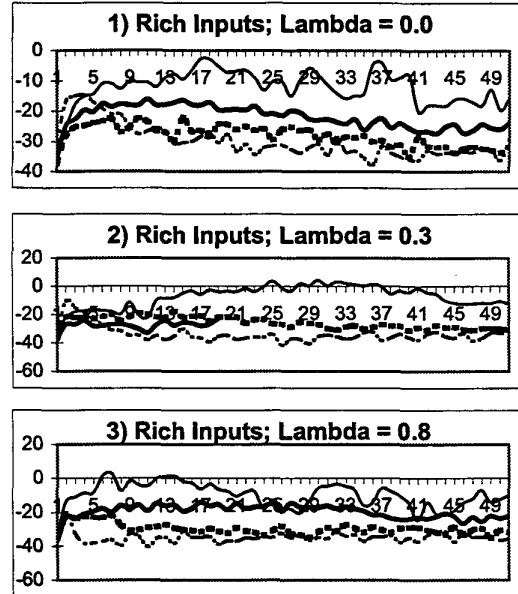


Figure 3: Comparisons of the learning curves with rich input representations. The y-axis represents average of (Network_score - Wally_score) in test games and x-axis represents the training cycles upto 50,000 games. The legends are: dark lines = 3 layer net; lighter lines = 2 layer net; solid = self play; dashed = opponent play.

Comments on the Single Network Performance: The two clear observations are: (1) networks with rich input features played well; and, (2) the learning by self-play is better than that against Wally. The 2-layer network with rich inputs and $\lambda = 0.3$ showed the best performance among the experiments. For $\lambda=0.8$, the network improved fast but it could not sustain the performance on further training. It could be suggested to start with a high λ value and then discount its value along the training cycle. An unwarranted characteristic of the learning curves in Fig. 3 is that the network performances dropped after cycles of consistent improvement. This may be explained in terms of stability-plasticity dilemma: the networks learned their weights from on-line training games and it may forgot a few good feature when it sees a new game. The slow rise and fall of learning curves in the succeeding training shows re-learning of the old features or learning of new features.

For the rest of the experiments in 9x9 Go, we considered only rich input representations and self-play during the training cycles.

Mixture of Neural Net Experts Architecture

The public domain program, Wally, can play at most 4-5 move variations at any given position and the starting move (the very first move) strongly dictates Wally's line of play. For 9x9 Go, we can train 81 critic networks, where the critic (x, y) is responsible to evaluate all the positions in a game when that game starts with a black move at point (x, y) . The authors trained 81 such critic networks using the adaptive critic design (ACD) [3] and the system of critics were able to collectively defeat Wally in every test games. As a logical extension of that experiment, the authors combined TD(λ) method and mixture of experts techniques [9], [10], to the game of Go. We designed two learning architectures for the critic networks to test the mixture of experts algorithms. They are: (1) Hybrid Mixture of Experts (HME) network [9], and (2) Meta-Pi network [10]. The HME and Meta-Pi learning rules are described in the Appendix.

We applied the TD learning on the HME and Meta-Pi system equations to learn 9x9 Go evaluation function using 1-level 3-expert networks as shown in Figure 4. Two gate strategies for the propagate-node are used: (1) cooperating gate: the network output is given as $y = g_1y_1 + g_2y_2 + g_3y_3$, where y_i and g_i are the outputs for expert I and gate-node I, respectively; and, (2) winner takes all: the network output is given as $y = y_k$ when $g_k = \max(g_i)$ for all I).

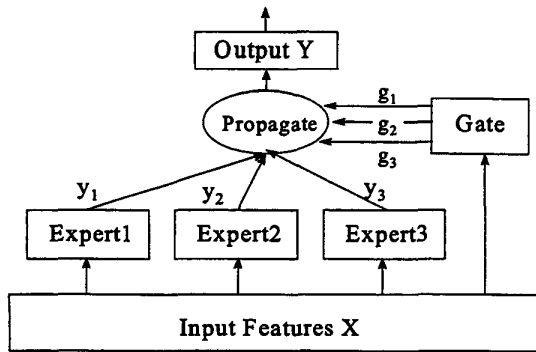


Figure 4: The 1-level 3-channel mixture of expert network. The experts and the gate are neural network and see the same input vector. The propagate-node forms the network output using the gate and the expert outputs, respectively.

We used rich input features to learn the experts and the gate functions, respectively, 2-layer and 3-layer feedforward neural networks were used. The experiments with the 3-layer neural networks showed that the networks were too slow to come out of the initial

ignorance. The best performance for the mixture of expert network with 3-layer MLPs used in the expert and in the gate was achieved by the HME training method with a cooperating gate function (only 9% win against Wally and the margin of victory improved from -52.0 to -21.1). The learning curves for HME and Meta-Pi networks with architecture with 2-layer MLPs forming the experts and the gate are shown in Figure 5. Table 1 shows the performance of the best networks against Wally in test games.

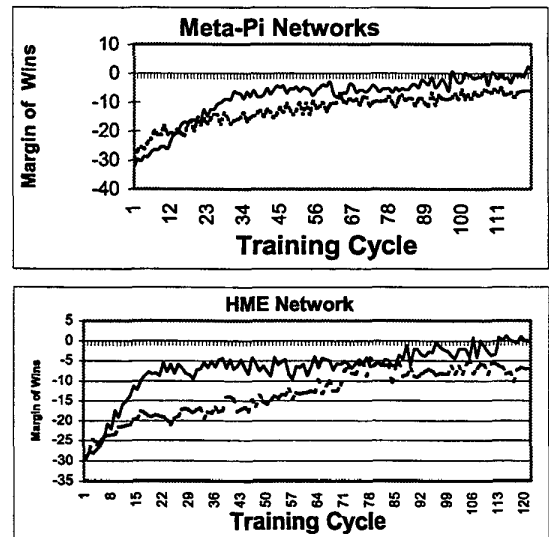


Figure 5: Comparisons of the learning curve for a 3-expert and 1-level HME network. The experts and the gate are formed with 2-layer MLP and they used rich input features (each has 247 input nodes). Each unit of the training cycle measures 500 games. Legends: solid = cooperating gate; dashed = winner takes all gate.

Table 1: Performance of 1-Level 3-Experts HME and Meta-Pi Critics. The Performances of the cooperating gating networks are very good. The HME network surpassed the strength of Wally.

Expert Architecture →	HME		Meta-Pi	
Gate function	WTA	Coop	WTA	Coop
1. Network wins in a test series in %	34	55	36	46.5
2. Av. margin of victory by network	-5.55	2.28	-5.52	0.75

Conclusion

The cooperative gates in the HME and Meta-Pi systems with three 2-layer neural-network experts performed very well. These systems deserve further considerations. One advantage of the mixture of expert networks over a single feedforward neural network is that, even with 2-layer MLPs they can learn non-linear mapping because of the gate action.

Acknowledgment

We gratefully acknowledge financial support from the following sources: (i) Texas Tech Center for Applied Automation and Research, (ii) National Science Foundation Neuroengineering Program and Knowledge and Cognitive Systems Program, (iii) Army Research Office AASERT Grant Program, (iv) NSF Research Equipment Grant Program.

Appendix

The HME Network: The HME learning for the above 1-level 3-expert system is defined by the following equations [9]:

The log likelihood function,

$$L(\Theta, x) = \ln \left\{ \sum_i g_i e^{-\frac{1}{2}(d-y_i)^2} \right\}; \quad (6)$$

the posterior probability h_i for the a priori probability g_i ,

$$h_i = \frac{g_i e^{-\frac{1}{2}(d-y_i)^2}}{\sum_i g_i e^{-\frac{1}{2}(d-y_i)^2}}. \quad (7)$$

The learning algorithm maximizes the log likelihood function using posterior probabilities. The partial derivatives of L with respect to i^{th} expert-network output and gate output, respectively, are

$$\frac{\delta L(\Theta, x)}{\delta y_i} = h_i (d - y_i); \quad (8)$$

$$\frac{\delta L(\Theta, x)}{\delta s_i} = h_i - g_i. \quad (9)$$

The Meta-Pi Approach: The Meta-Pi network has the same structure as given in Figure 4. The Meta-Pi system equations for the architecture in Fig. 6 are [10]:

The output of the architecture,

$$y = \sum_i y_i g_i; \quad (10)$$

and, Meta-Pi error,

$$E = \frac{1}{2}(d - y)^2 = \frac{1}{2} \left(d - \sum_i g_i y_i \right)^2; \quad (11)$$

the partial derivative of E with respect to the output of the i^{th} expert,

$$\frac{\delta E}{\delta y_i} = \frac{\delta E}{\delta y} \frac{\delta y}{\delta y_i} = -(d - y) g_i; \quad (12)$$

the partial error derivative with respect to the i^{th} output of the gating network,

$$\frac{\delta E}{\delta s_i} = \frac{\delta E}{\delta y} \frac{\delta y}{\delta s_i} = \frac{-(d - y)(y_i - y)}{\sum_j s_j}. \quad (13)$$

References

1. Arthur Smith, *The game of Go*, Charles Tuttle Co., Tokyo, Japan, 1956.
2. G. Tesauro, "Practical Issues in temporal difference learning," *Machine learning*, No. 8, 1992, pp. 257-278.
3. R. Zaman and D. Prokhorov and D. Wunsch, "Adaptive Critic Design in Learning to play Game of Go," *Proc. Of ICNN*, Houston, vol. 1, pp. 1-4, 1997.
4. 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.
5. M. Enzenberger, "The Integration of A-Priori Knowledge into a Go Playing Neural Network," from <http://cgl.ucsf.edu/go/programs/NeuroGo.html>, 1996.
6. S. Sutton, "Learning to predict by the method of temporal differences," *Machine learning*, No. 3, 1988, pp. 9-44.
7. Bill Newman, "Wally - a Simple Minded Go-program," Shareware Go program available by anonymous ftp from <ftp://imageek.york.cuny.edu/nngs/Go/comp/>.
8. M. Raonak-Uz-Zaman, Applications of neural Networks in Computer Go, Ph.D. dissertation, Texas Tech University, 1998.
9. M. Jordan and R. Jacobs, "Adaptive Mixtures of Local Experts," *Neural Computation*, vol. 3, no. 1, 1991.
10. J. B. Hampshire and A. Waibel, "The Meta-Pi Network: Building Distributed Knowledge Representations for Robust Pattern Recognition," *Technical report, CMU-CS-89-166*, Carnegie Mellon University, Pittsburg, 1989.