Machine Learning Approach in Game Go

Course Project
CS365 - Artificial Intelligence Programming

Rohit Gurjar

Guided By : Prof. Amitabha Mukerjee
Dept of Computer Science and Engineering
IIT Kanpur - 208 016
INDIA

April 15, 2008

ABSTRACT

In this project I have implemented a Go player using a Machine Learning approach. The algorithm used is based on the ideas of Temporal Difference Learning[1]. This approach enables the player to learn by playing with itself. The idea of incremental learning was introduced by Sutton in 1988, and it had yielded successful results for Backgammon(another complex game). The approach does not seem directly applicable in GO. I modified the basic idea of giving the same reward for each state occuring in a game to that of exponentially decreasing rewards from the last state to the begining state. The learned player wins almost all games against a random player (which generates random moves). Against GNU Go its performance was not good becuase it had not been trained with KO rule(due to complexities involved), which plays a very important role. But the pattern of its games showed that it learnt quite a few strategies. The results show that this approach can work in Go but it will need a large number of games for learning and also training with the KO rule.

Introduction

The game Go is a board game originated in China around 2000 years ago. The game is perhaps the oldest board game in the world and mostly popular in East Asia. This game has very simple rules, and we can learn them in a very little time, but is known for strategically being very complex. To know more about the rules and the history of the game visit $http://en.wikipedia.org/wiki/Go_(board\_game)$. A lot of work has been done towards creating a computer go player. But we are still standing far away from the success. Current best programs are ranked only 1-3 kyu, which is equivalent to a amateur human player. The standard AI techniques do not work well in the case of Go because of these factors-

Large board size The large board(19X19) is one of the main reason that a strong program cannot be created. At any point in the game on an average there are around 200-300 possible moves for a player exists. It makes the search space very huge.

Increasing Complexity In most of the board games as game proceeds the pieces disappear from the board which reduces the complexity of the game. On contrary in Go complexity increases further as the games goes on.

Absence of Material advantage In capture-based games (like chess), a position can often be evaluated relatively easily by calculating who has a material advantage. In Go, there is often no easy way to evaluate a position. There is no concept of material advantage in Go. Even if a result could be evaluated locally, the quality of the result also depends on the whole-board position.

These difficulties gives motivation to go towards machine learning approach. In this appraoch the aim is to learn a Evaluation function, which gives a real number for each state(Board position), Which shows the probability of winning the game from this state. The player moves according to this evaluation function, it moves to the state which has the best value. The player learns this evaluation function by self play.

Approach

In this project I have only dealt with 9X9 board. The approach used have ideas similar to Temporal Difference Learning. It uses Neural Network Technique. First I will describe the Neural Network technique then ellaborate more about the approach.

Neural Network is a technique used in supervised learning, where evaluation function at many points(states) is already known. These values are used for approximating the function over whole state space. Neural Network is basically a computational model of brain. It consists of many perceptrons connected to each other. One perceptron takes some input values and results an output value. Each perceptron has some weights,

for calculating the output perceptron takes the weighted sum of the inputs and transfers it to a transfer function. The transfer function I used is the following sigmoid function.

$$y = \frac{1-\exp^{-ax}}{1+\exp^{-ax}}$$

The above transfer function is used because I wanted the output value to be from -1 to 1. 1 corresponds to very high probability of winning while -1 corresponds to a very low probability of winning and 0 corresponds to half probability of winning.

Neural Network has an input layer, an output layer and some hidden layers of perceptrons(shown in fig. 1). Input layer consists all the inputs (some representation of the state, for example in Go inputs can be 81 values from {-1,0,1}, where -1 represents white stone, 1 represents black stone and 0 represents empty intersection). The inner layer nodes are connected to the first hidden layer and work as inputs to the hidden layer nodes. Similarly, outputs of the first hidden layer go to the inputs of the next hidden layer and so on. Outputs of the last hidden layer go to the output layer nodes as input. Finally the output of the output layer(which has one node) is the output of the neural network.

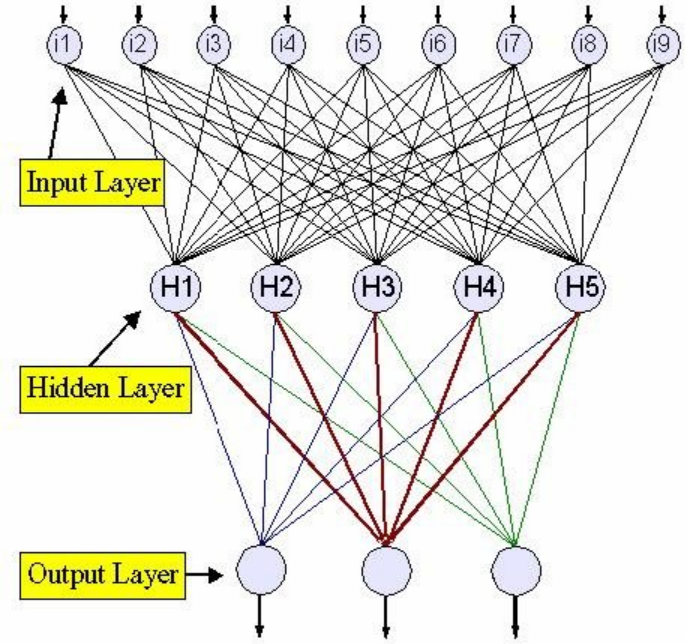

Figure 1: Layered Architecture of Neural Networks [2]

Now, there is a desired output(target) for some states. The neural network gives a real value for each state dependent on weights of each perceptron. The weights are adjusted such that the coming output will shift towards the desired output. Following expression shows the change in weights.

$$\Delta w = \alpha(Z - Y)\nabla_w Y$$

where Δw is the change in the weight vector, α is the learning rate, Y is actual value corresponding to the state and Z is the target value. Such passes through many states with some desired output will give the approximate evaluation function. Learning was done in two phases.

Initial Learning

Initially all the weights are chosen randomly. Then, a perfectly symmetric state is randomly generated and the target value for this state was set to be zero. The Neural Network was made learned through a large number of such states with target value zero. The number of states generated for initial learning was 1000000. But the performance against a random player did not improve much after initial learning.

Actual Learning

In actual learning player learned by playing with itself. At each step of one game player evaluates all the possible next states using the neural net and then move to the state with the best value. After playing one such game final score is counted(using chinese rule). Positive score means win and negative means lose. This score is mapped to a value between -1 and 1 (a map similar to sigmoid function).

$$z = \frac{1-\exp^{-0.35*x}}{1+\exp^{-0.35*x}}$$

where x is the final score. The value z is set to be target value for the last state and then the weights of neural network are adjusted accordingly. The target value for the second last state is set to be $0.98 \times z$. Similarly, for the intermediate steps target value is exponentially decreased. For the winning side target value is positive and for the loosing side target value is set to be negative. So, the final weight adjustment can be given by following expression.

$$\Delta w = \alpha(f_t z - Y_t)\nabla_w Y_t$$

where Y_t is value corresponding to the state at time t, f_t is the function for exponential decay of target value. In this project A 3-layered neural network is used, one input layer, one hidden layer and one output layer. Number of nodes in input layer is 83, 81 of which are for representing 81 intersections of the board and one is for score corresponding to the state(using chinese rules) and one is for representing game length. The node for game length is important because playing stragtegy at begining differs from the end game. This idea I took from the paper "A machine learning approach to computer go" [2]. The hidden layer is twice the size of input layer i.e. it has 166 nodes. Output layer has one node.

Results

After 100 games of self playing the player has aquired a good level of learning. It is defeating a random player most of the times. As the number of learning games increases, its level of playing is increasing. The table below shows the results against a random player.

	No. of games	No. of wins(Out of 100)	Total Score
1	0 (random weights)	15	-2272
2	100	87	1862
3	1000	94	2043
4	2000	97	2278
5	3000	100	2532

Total score is sum of scores of the all the 100 games.

Results against GNU Go

I manually tested the player against GNU go(only one game). In the game GNU go(white) won with 9 points margin. The final board position is shown in the fig 2. It is clear that the defeat would have been of very large margin, if the GNU go would have continued the game but it passed. So, the performance observed was very poor. But some interesting observations were also noted. I observed a pattern that the player tries to make a group with two eyes but the opponent never let him succeed. In fig 3, in the right top corner it(black) was trying to make a group with two eye but failed. So, finally it can be concluded that the player is learning substantial amount of game, and the approach used can work with some appropriate parameters and it will need a large number of games to learn a good strategy of the game.

Further Improvements

A big improvement can be done by make the player learn against GNU Go instead of itself. Also I have not implemented scoring and ko rule properly because of complexities involved. With proper implementation of these parts of the game the performance will surely improve.

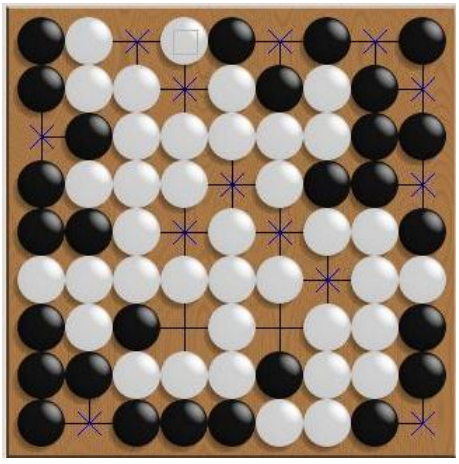

Figure 2: Final Board position against GNU go

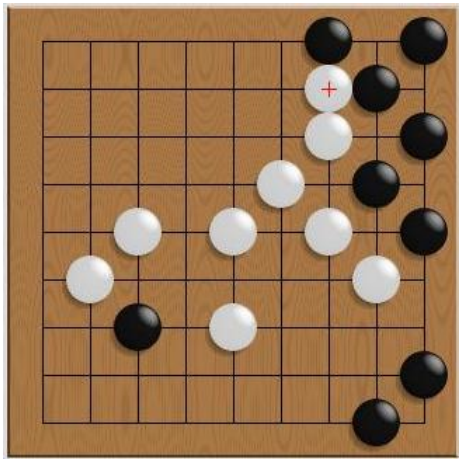

Figure 3: An intermediate state

References

[1] Richard S. Sutton, "Learning to predict by methods of temporal differences", 1988.

[2] Jeffrey Bagdis, "A Machine-Learning approach to computer Go", 2007.