David Leblanc

Assignment #1

CS773C Machine Intelligence
09/05/2012

Goal:

Design a Connect4 game program which can play against a human. The "AI" should use a minimax tree with alpha-beta pruning, and must have an adjustable ply depth. This program should ultimately be challenging to beat. The weights used for the static evaluator should be trained instead of arbitrarily chosen.

Design:

The design is the same as the previous assignment. We want to train the weights for the player by facing off the previously hand-tuned player against a player with randomized initial weights. As the game plays itself, random game states are grabbed and the weights of the player are updated at the end of a game. The players keep playing each other until the error of the LMS is small, or until there is little to no change in the updated weights, or if the trained player manages to defeat the hand-tuned player.

Strategy:

There are a few aspects to the strategy for training this player. First, as stated above, random board states are grabbed as players are playing, and the trained player is updated at the end of the game. Updating at each board state appeared to cause problems. Second, the LMS error which should be minimized is given as:

$$error(b) = V_{train}(b) - V'(b)$$

One thing to consider is that a board position b that is good for one player should be bad for the other player. This essentially means that the result of evaluating b for one player will be a positive value and for the other negative. So intuitively, the equation would in fact be,

$$error(b) = V_{train}(b) + V'(b)$$

Then the weights should be updated using the following formula:

$$w_i = w_i + c \cdot f_i \cdot error(b)$$

The weights are updated at the end of a game. Once the weights converge on a solution, the optimal trained player weights have been found. Finally, an important step in the training is that the weights should be normalized. The way the weights are normalized is by bounding the weight vector values in range -1 to 1. Essentially, dividing each weight by the maximum absolute weight.

Static Evaluator:

As in assignment 0, the main features used for the static evaluator to describe the game state are as follows:

- Count of groups of two (for both red and black)
- Count of groups of three (red and black)

- A successful group of four (red and black)
- Average distance from the center column of the board (red and black)

The groups of two and three are only counted if it is possible to make a straight line of four tokens using that group. In other words, for example, if black tokens are found between, or at the end of a line of red tokens, or if the line is on the edge of the board, this group is not accounted for. The average distance from center essentially means the closer a token is from the center, the higher it's score is. Tokens further from the center column (on the edge of the board) yields a lower score.

The state evaluation is done as follows:

$$f(GameState) = w_1 \cdot c_{2R} + w_2 \cdot c_{2B} + w_3 \cdot c_{3R} + w_4 \cdot c_{3B} + w_5 \cdot c_{4R} + w_6 \cdot c_{4B} + w_7 \cdot d_{c,R} + w_8 \cdot d_{c,B}$$

Where w are the weights, c the counts for groups of two, three, and four, d the distance from center, and R/B stands for red/black. In the last assignment, the weights w were constant, fixed, and manually chosen. The values for the weight have been chosen somewhat arbitrarily, based on experience playing the game. The normalized weights used are as follows (assuming the "AI" player is blacl):

$$w_1 = 0.072, w_2 = -0.09, w_3 = 0.091, w_4 = -0.273, w_5 = 0.164, w_6 = -1.0, w_7 = 0.091, w_8 = -0.091$$

As shown above, the fourth and sixth weights are heavily "punished" which makes sense considering that is the count of triplets and win condition for the opponent, black. The fifth weight (the winning case for red) is not as favored, since the goal of these weights is to deny the opponent a victory. As mentioned previously, these weights are not the most exact and unique weights, they are simply designed for this type of strategy.

Results:

After running the LMS training method on the red player, the weights calculated manages to tie the black player with the hand-tuned weights. The resulting weights trained for the player are as follows:

$$w_1 = -0.414265, w_2 = 0.121151, w_3 = -0.132171, w_4 = -0.0326753,$$
$$w_5 = -1, w_6 = 0.239334, w_7 = -0.259813, w_8 = -0.0984945$$

The resulting weights appear to result in a more defensive player. This player will tie the game regardless of it plays first or second. This seems reasonable as the way it was trained was to minimize the error from the hand-tuned player. This in practice results in weights that will not try to win, but essentially block the hand-tuned player from winning. This is arguably an improvement over the hand-tuned player since it is unable to beat it.