TicToeTac - Learning TicTacToe with the LMS algorithm

Johann Wagner, 212276
Johannes Wünsche, 211720
Marten Wallewein-Eising, 212277
Paul Stang,

5. November 2017

1 Overview

This is the documentation for our implementation of the LMS Algorithm for TicTacToe. The order of the named steps roughly represents the planned arrangement of execution.

2 Move decision

Each turn the player calculates the board value which will exist after making a move, for all positions still not occupied by any of the players. Afterwards the one with the heighest score will be returned as confirmed move.

3 LMS Parameters

Our function which will be approximated is defined as $f : Board \rightarrow Score$, so we calculate a score of the current board state depending on the current calculated weights $w_0, ..., w_4$. We decided to use the following parameters for our LMS function.

- x_1: Count of own rows, which can be completed
- x_2: Count of enemy rows, which can be completed
- x_3: Count of min marks required to fill a row
- x_4: Count of min enemy marks required to fill a row

These parameters are extracted from the board for every possible move that can be made to estimate the best move regarding the current board state. We did some calculation with example boards to set the weights to useful initial values to avoid starting on complete wrong weights, because of this uncertainty the learning rate was chosen very small with $\mu = 0.001$.

4 Obtaining the parameters

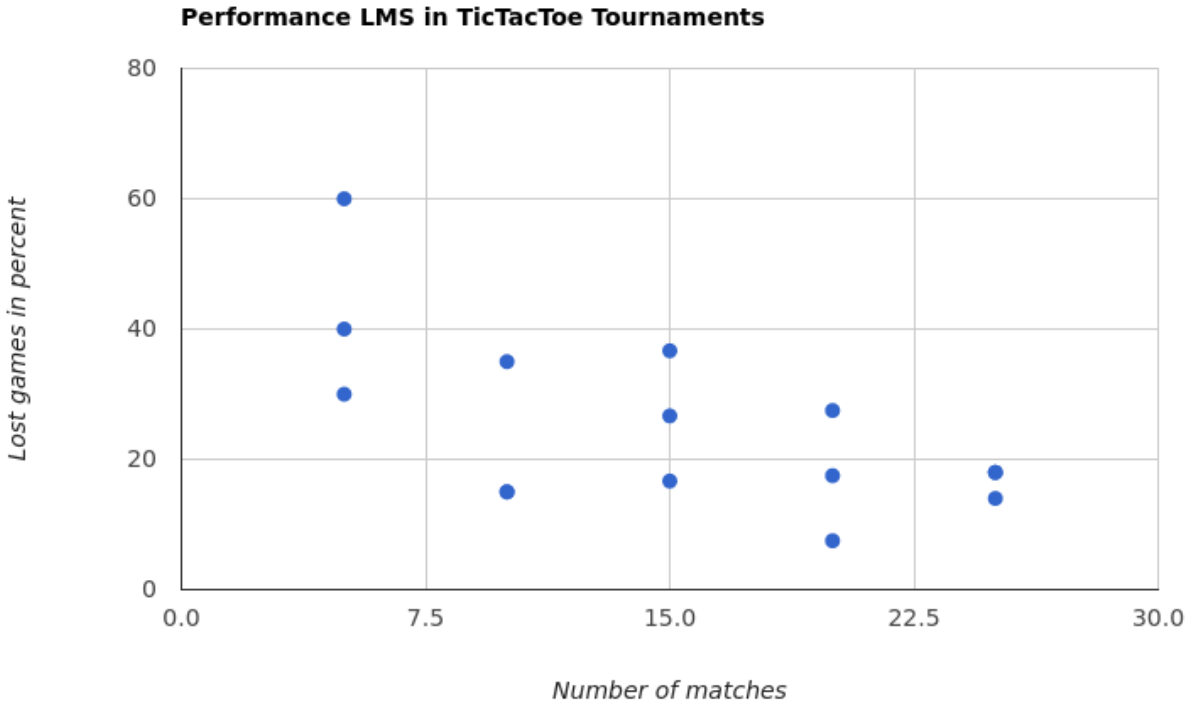
In order to enable our LMS to work correctly the algorithm needs to catch the board parameters everytime it needs to be calculated. This is done with a rather simple brute force check for all rows, and columns and vertical equivalents, which then returns an array consisting of the desired values.

5 Learning

Our player learns each time a match is finished comparing the calculated board score to defined scores for win (100), draw(50) and lose(-100), in regard to the exisiting error descriped by the absolute difference between the end state and our calculated value for the baord. We thought about learning in each step comparing the current board to the last board, but we can not be sure if the calculation of our last board was close to the target function in the first matches.

6 Results

All tests were conducted with the starting weights $\{0, 1, 1, 1, 1\}$.



As seen in the chart after a few training matches the algorithm lowers its losing rate, this can be traced back to the changing weights, which are recalculated after every game. We can also see that the losing rate gets lowered less and less the more games have been completed, leading to the assumption that while this algorithm can increase the win rate, it can not raise it to 100%, but due to the random nature of the competing player this was to be expected.

6.1 Grading of the solution

While the algorithm certainly improves the player, this is not the optimal solution. In matches against the smart player it still shows deficiencies, which will lead to the player defeat. This may be undone by further training or improvement of the algorith itself.