Reinforcement Learning in Connect 4

Michael Yura

2007-2008

Introduction:

Although an AI is often thought of as being only as intelligent as its programmer, this is not exactly the case; this project will attempt to create an dynamically learning Machine Learner for Connect 4 by using supervised reinforcement learning, with each Learner saving the way that it will play into text files, each directing the way that it will play for a given board layout. Several variations of these “ML”s will be programmed with differing algorithms, which will play against one another thousands of times, and the results will be recorded to see which variation is able to adapt the best.

Background:

I have created a dynamically learning ML (Machine Learner) for Connect 4. These ML's will learn through reinforcement learning. If they are successful (and win the game), they will do what they did more often; conversely, if they are not successful (and lose the game), they will do what they did less often.

Through this project, I hope to learn how fast and to what quality reinforcement learning allows for the learning of a simple game; these methods can hopefully be extended to other, more complex tasks for machines to learn.

Connect 4 has already been solved by James D. Allen and Victor Allis, and their strategies have been outlined in Allis's *A Knowledge-based Approach of Connect-Four*. The use of algorithms in reinforcement learning has also been explored in Ronald J. Williams's *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*.

Procedures:

I have programmed the Connect 4 game itself, as well as created an “ML” (Machine Learner) abstract class that other ML's will be based upon, with methods to load save its board data. This ML appends all of its board data into a single text file and saves the corresponding probability data its own smaller file, which is rewritten after the ML plays a game.

There are four ML types: NONE (NON), CONSTANT (CON), LINEAR (LIN), and LOGARITHMIC (LOG). Each variation multiplies its values, *P*, by:

$P=P*p$

if it wins a game and:

$P=P/p$

if it loses.

The variable *p* is determined by the ML’s type.

NONE: $p=1$

CONSTANT: $p=k$

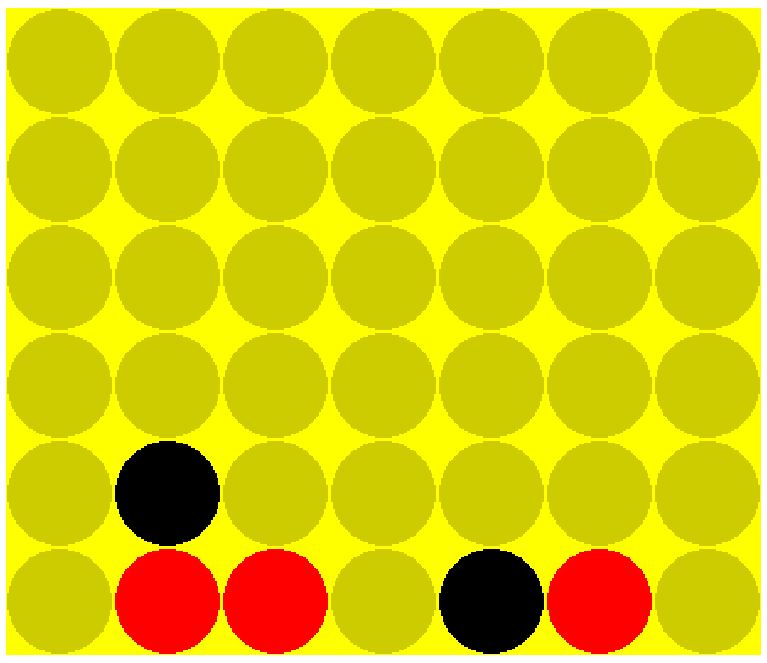
LINEAR: $p=k*(t+1)$

LOGARITHMIC: $p=\ln(T)+1-\ln(-t+T)$

Where *t* is the number of the current turn, *T* is the total number of turns, and *k* is an input number.

Each ML played 10,000 games against each other ML type, and the results of the games were recorded and graphed.

A board of:

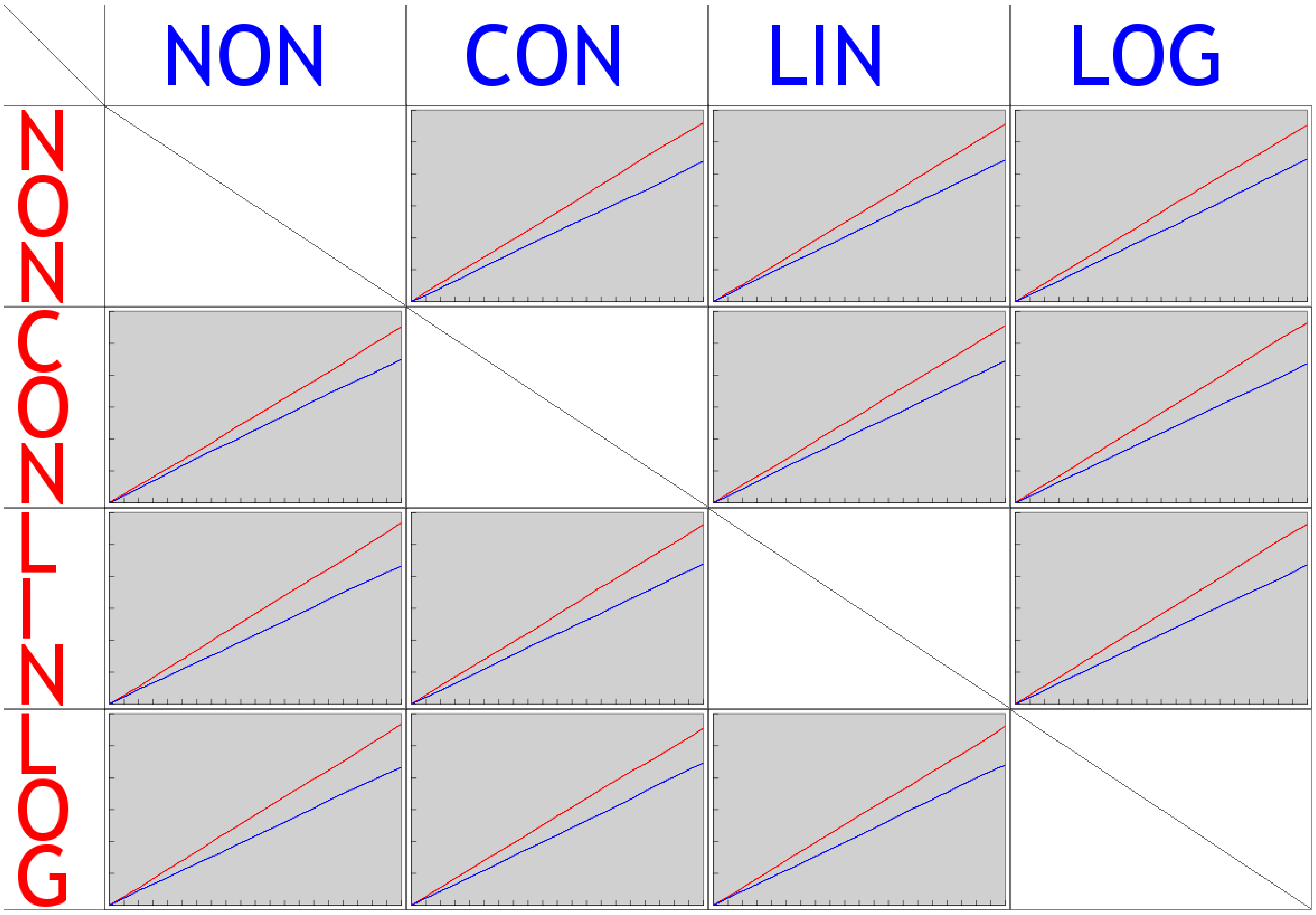


Would be represented in the board data file as:

[0,0;0][0,1;0][0,2;0][0,3;0][0,4;0][0,5;0][1,0;1]
[1,1;2][1,2;0][1,3;0][1,4;0][1,5;0][2,0;1][2,1;0]
[2,2;0][2,3;0][2,4;0][2,5;0][3,0;0][3,1;0][3,2;0]
[3,3;0][3,4;0][3,5;0][4,0;2][4,1;0][4,2;0][4,3;0]
[4,4;0][4,5;0][5,0;1][5,1;0][5,2;0][5,3;0][5,4;0]
[5,5;0][6,0;0][6,1;0][6,2;0][6,3;0][6,4;0][6,5;0]

Similarly, a probability data file of:
[94.0,15.6,77.2,92.8,100.0,43.3,0.1,]

Would represent a:
94.0/423.0 (22.22%) chance of placing in Column 0
15.6/423.0 (3.69%) chance of placing in Column 1
77.2/423.0 (18.25%) chance of placing in Column 2
92.8/423.0 (21.94%) chance of placing in Column 3
100.0/423.0 (23.64%) chance of placing in Column 4
43.3/423.0 (10.27%) chance of placing in Column 5
0.1/423.0 (0.02%) chance of placing in Column 6



These are the graphs of the test results; the x-axis is the games played, and the y-axis is the number of games won. The red line represents the results for player 1, and the blue line represents player 2. Each row of the grid represents the ML type of player 1, and each column represents the ML of player 2.

Results and Conclusions:

Although each ML played against every other ML thousands of times, there was no significant change in how well each ML could do. In every instance, the first player to place a piece had an advantage, and won a significantly greater percentage of the time than the player who acted second. Any positive self-improvement that may have occurred was not evident in the results of the tests.

The ML algorithms are designed so that they can significantly improve against a static opponent, one that uses the same strategy every time - the Mls cannot adapt quickly and correctly enough to face a human player or another ML, whose game strategy is determined randomly.

This failure to significantly improve is not merely the fault of the learning concept, but of the situation that the Mls were put into as well. More complex algorithms such as LINEAR and LOGARITHMIC depend on the fact that the opposition’s moves were supposed to be in its favor, so that the algorithm can “know” what to avoid and what to repeat; when facing a randomly-playing ML, the algorithm is rendered useless.