A Connect Four Heuristic Learning Machine

Bryan Kern
Computer Science Department
State University of New York at Oswego

Abstract

In this paper there are a number of issues brought up. The first being the idea of artificial intelligence, while leading into machine learning. This paper outlines how one goes about incorporating artificial intelligence into research and how we, as humans, can learn from artificial agents. The project task at hand was to create a program that incorporates artificial intelligence, especially the aspect of machine learning. The famous child's game Connect Four was used in this project to see whether or not an artificial agent can create heuristics, and in turn use them to win at the game.

Keywords: Connect Four, heuristic, LISP, machine learning, symbolic artificial intelligence, perfect play

Introduction

Artificial intelligence has been something of awe and wonder throughout history. It has not been until recently though that people have been actually trying to figure out ways to utilize artificial intelligence to see whether or not something artificial can learn. Turing thought of a way to try and recreate a human child's brain using artificial means (Turing, 1950), while today people are using artificial intelligence to help aid in doing jobs that humans do not have a capacity for, or that they do not want to do. But, this paper takes a more academic learning approach to artificial intelligence.

This paper looked at how heuristics are incorporated with a machine learning agent to play the famous board game Connect Four (Wexler, 1974). The first section of this paper dealt with background information on machine learning, looking at heuristics used in LISP. The next section dealt with related research to the topic at hand. The third section dealt with the approach to how the machine was built. The fifth section discussed knowledge representation with respects to symbolic AI. The sixth section dealt with a conceptual idea of how the program works. The seventh and eighth sections are the results and discussion, respectively. The last two sections involved future research and a conclusion.

Background

This project's aim was to create a heuristic learning machine to be able to play the famous board game Connect Four. The first thing to discuss is what a heuristic actually is when it comes to machine learning. The next is to discuss how to program Connect Four to be able to do this, and certain issues that must be recognized when one tries to program any type of artificial agent (van den Herik, et. al., 2002).

The project used the programming language LISP to model Connect Four. Lisp was created by John McCarthy in the late 1950's and was used to process symbolic data (Siebel, 2005). LISP stands for LISTIC Processing; it's what the language is used for, and very good at it too (Siebel, 2005). As for machine learning and heuristics, both of these have strong roots in artificial intelligence from it's inception to modern day AI (Simon & Newell, 1958; Romanycia & Pelletier, 1985; Clune, 2007). Both of these topics will be brought up throughout the paper.
Related Research

There is a lot of research that deals with relating heuristics and machine learning. Simon and Newell had ideas about machine learning back in the late 1950's, which were grounded in heuristics (1958). As time went on more and more research was done in the field of machine learning with respects to heuristics. Romanycia and Pelletier described what a heuristic was in 1985 and Clune described how to use heuristics with respects to general game playing in 2007. The idea of heuristics in machine learning and AI is not a new topic, but taking past ideas and testing them using modern technology is what is happening today.

As for relating machine learning to Connect Four, Victor Allis found a perfect play algorithm to be able to win the game every time (Allis, 1988). Allis found that there the game is able to be beaten by who ever goes first.

Approach

The project needed to be broken down in ten different tasks, each of which being something that built upon the previous task. The hope in the end was to have something of substance that was in respects to symbolic artificial intelligence. The aim for this project was to create a machine that can create heuristics based on past wins that it has come across during gameplay. So, before one can create this machine a number of different things had to occur first. After the initial game was built, in which a human could play a human, a random machine was created. This was to show some type of machine was able to play the game, but maybe not successfully. The next step was to create a heuristic machine that has a given set of rules programmed into it to be able to play a human, a random machine and an identical heuristic machine. The last step was to find a way to create a heuristic machine that could play a given number of games, and based upon statistics that it has gathered, be able to take all past wins and put them into some type of knowledge base. In turn, this machine would be constantly able to gather more and more wins in hope of being able to learn how to play the game to win.

This project also has at some level pattern recognition problems associated with it. Hofstadter (1995) has an entire chapter in his book *Fluid Concepts and Creative Analogies. Computer Models of the Fundamental Mechanisms of Thought* that deals with pattern recognition and how one can go about programming in pattern recognition into machines.

Knowledge Representation

The major knowledge representation that was ascribed to this project was the use of a knowledge-base to store the winning heuristics for the machine to later be able to use.

Program Abstraction

The program was created in LISP and has a couple of different components attached to it. I will go through each major part of how the game was created in LISP.

Chips and Board

Both the chips and the board in the game were built using the object-oriented portion of LISP (CLOS). The reason that the object-orientation was used was the fact that for this assignment it felt as though every chip and place on the board should be their own object. Meaning, that everything in the game should be able to be referenced some how. This made programming both easy for some aspects, but also a bit more challenging for others. Take for instance a method that wants to display all of the
available-locations on the board. Well, there are only seven available-locations at any given time, so figuring out how to do this with 42 individual spots on the board was a worthwhile task. Below is an example of what the board looks like at the initial stages of the game. The ‘+’ indicates a blank spot on the board. The numbers 1-7 denote columns and the letters A – F indicate rows. The second example indicates what the play of the game looks like with a black chip being denoted at 'B' and a red chip being denoted at 'R'.

Figure 1:
Blank Board:

```
  1 2 3 4 5 6 7
A + + + + + + +
B + + + + + + +
C + + + + + + +
D + + + + + + +
E + + + + + + +
F + + + + + + +
```

Figure 2:
Board in play:

```
  1 2 3 4 5 6 7
A + + + + + + +
B + + + + + + +
C + B + + + + +
D + R + + + + +
E + B + + + + +
F + B + + + + +
```

**Human Representation**

The human versus human game (and any of the different types of games for that matter) is set up in such a way that regardless of color, which was programmed so that the first person to enter their name for the game gets red and the other player gets black as their respected chip color for the game (NOTE: this is most likely going to change depending on the type of agent playing the game, just for the fact of making it easier to keep track of when it comes to seeing who goes first when looking at lots and lots of results from heuristic machine runs). But, for the human versus human (and human versus random / random versus random) this does not really matter.

Game play goes as follows. As soon as the game is initiated a prompt comes up for the first players name (which will be mapped to the RED chip) and then the second players name is prompted (in turn being mapped to the BLACK chip). Then, a prompt comes up asking who will go first. From here, who ever chose to go first will be asked which column they want to put their chip in. This is denoted by an initial list of (F1 F2 F3 F4 F5 F6 F7). Once the player makes a move, the list will update according to what available-locations are still there. Looking at figure 2 above, the list for who ever went first would be (F1 B2 F3 F4 F5 F6 F7). The player has to choose from on of these available-locations. If they do not, then they will be prompted to choose a new space. The game iterates through until there is a winner. Which, if this is the case the winning player will be congratulated and the game will end. All 69 of the possible wins are hardcoded into the system.
**Random Machine**

The random machine is similar to the human gameplay, with two minor differences. The first being that when the machine is prompted to choose a location, it will randomly pick from the available-location list. Then, for the random versus random game, when prompted for who will go first, random player one will always go first.

**Heuristic Machine**

...

**Heuristic Learning Machine**

...

**Results**

...

**Discussion**

...

**Future Work**

It would be interesting to take this connect four heuristic machine and see if it will learn the perfect play method that was discovered (Allis, 1988). Then, be able to take the machine that learned how the perfect play and play it against another machine that has had the perfect play hardcoded into it, and see which one fairs better. That is, alternating first player between the two machines since perfect play requires the first player to go first in order to be successful at perfect play (Allis, 1988). There are also programs that take a connectionist approach to Connect Four, which would be interesting to look at (Schneider, et. al., 2002).

**Conclusion**

...

**References**


Schneider, M.O., Rosa, J.L.S. (2002). Neural connect 4 – a connectionist approach to the game. Brazilian Symposium on Neural Networks.


