ARTIFICIAL INTELLIGENCE: AN APPLICATION OF REINFORCEMENT LEARNING

Michael Darmousseh

Artificial Intelligence (AI) has advanced quickly through the decades from the first programs to play Chess to modern robots that can perform complex tasks liking learning to walk effectively. The evolution of board games is an interesting and dynamic topic within the field of AI. Some of the advances have been the result of modernization of classical AI algorithms. Algorithms are simply methods of solving a problem. My project involves adapting a modern algorithm to a classical board game and comparing it to a classical algorithm. In order to explore this, I have chosen the game of Connect Four. Perhaps by using Reinforcement Learning (RL), a new program will be able to effectively play against a computer in the game of Connect Four. If this is possible, it might be possible to apply Reinforcement Learning to more complex games like Go or Chess or even to help provide more efficient to problems where the state space is tremendous.

Connect Four

Connect Four is a board game involving two players taking turns placing pieces onto a board in allowed locations. The game is won when four pieces in a row are of the same color. Because simple board games have been traditionally played using the Mini-max (a classical algorithm, see Russell and Norvig), it will likely work for Connect Four. In fact, as a previous project I applied using the Mini-max algorithm and won 95% of the 40 games it played against members of my class. Although this is an effective algorithm for board game AI, I believe that a more modern algorithm called Reinforcement Learning will be more effective and win more often than using the Mini-max algorithm.

Previous Research

The classical algorithm I used is called the Mini-max. This algorithm works by examining the next immediate moves and attempting to determine how likely the program is to win in that particular situation. To determine the likelihood of wining a situation it evaluates the board with using a specified method. This method can be simple such as "If I win the game, then I have 100% chance of winning, or if I lose then I have a 0% chance of winning". Sometimes, when this is not immediately known, an established method will try to approximate the value instead. If there is not enough information about the value of the chance of winning or if it is possible the value can be better approximated than for each of those moves, the algorithm will look at all of the possible counter-moves and examine how likely the opponent is to win. This process will continue back and forth until some maximum number of moves, or some time limit, is reached. The algorithm always assumes that each opponent will choose the best possible move. So the computer will attempt to maximize the minimum gain and minimize the maximum loss from the opponent's perspective hence the name Mini-max. This algorithm excels in situations where the problem is well defined, results are easily understood, and the maximum number of moves is very small. Faster computers and more memory drastically increase the performance of the program.

Reinforcement Learning

One of the many problems in Artificial Intelligence is machine learning. Teaching a computer program how to do something well or optimal has been a problem since computers first came out. In machine learning the computer program uses algorithms to

learn the optimal set of actions in a given situation. In supervised learning this usually means a human teacher giving positive or negative feedback as a means of teaching or even devising a table of states and actions for the specific problem. This methodology has been used extensively in chess. However, supervision comes at a cost. Occasionally, humans do not understand the best actions in a given situation even if they are able to perform them. For example, balancing a pole in a hand is a simple task for humans to learn, but almost impossible to explain to a computer. In response to this obstacle, unsupervised learning or unbiased learning has become increasingly more popular. Instead of trying to explain to a computer how to balance a pole, this type of learning allows the computer to learn it on its own by sensors receiving input from the environment and performing actions. More completely, "Reinforcement Learning is defined not by characterizing learning methods, but by characterizing a learning problem. Any method that is well suited to solving that problem, we consider to be a reinforcement learning method." (Sutton and Barto, 1998, p. 4) Reinforcement Learning is a type of machine learning in artificial intelligence used to optimize computer learning in certain environments without supervision. It was developed initially around 1979 and has evolved over time. This goal of this paper is to examine the history, theory, and applications of Reinforcement Learning and examine an example to understand it more completely.

Artificial Intelligence and Reinforcement Learning

Reinforcement Learning Problems tend to be vast and can cover many ranges of problems. In fact "Reinforcement Learning might be considered to encompass all of AI: an agent is placed in an environment and must learn to behave successfully therin". (Russell and Norvig, 2003, p. 764)

The Basic of Reinforcement Learning

The definition of Reinforcement learning could be summed up as a method of learning designed to perform the best actions in given environments, with the purpose of maximizing a reward value and most importantly without human supervision. The reward of taking an action does not always have to be immediate, in fact the reward may not happen until the final state. Also Reinforcement Learning must be able to predict the reward of any possible action in any given environment. The Reinforcement learning algorithm, in all of its forms, presents a solution to this problem that other types of learning have failed to achieve. The makings of a reinforcement problem include these basic elements: the agent, the agent's environment, a reward function, a value function, and a policy. The agent is the actor in an environment. Typically this represents a program, but this could represent any object not limited to robots, vehicles, or an entire computer itself. The environment of the agent is the place or conditions it must face. In chess the environment would be the board, in robotics perhaps the floor, in a traffic light system this would be all the conditions that apply like car collisions are not allowed. The reward function is the goal of the problem. This reward is numerical and represents the desirability of the state. The goal of the reinforcement program would be to maximize this reward. In chess, winning would be given a score of 1, and losing or tying would be given a score of 0. The rewards do not need to be immediate. In chess the reward is not achieved until someone has been checkmated, resigned, or declared a draw. Finally, the policy is the agent's way of learning to perform certain actions in given situations. The policy could be simply defined as a mapping of environments to actions. In learning to balance a pole this could be something like "If pole is leaning left by 20 degrees or more, move left 3 units".

A More Complete View of Reinforcement Learning

Reinforcement Learning can be applied to any number of artificial intelligence problems of any environment (fully observable or partially observable, deterministic or stochastic, sequential or episodic, static or dynamic, discrete or continuous, and single agent, or multi agent.) The only requirements of Reinforcement Learning are that the environment can be quantified or described in some fashion, a finite number of actions are possible in each state, and that a quantifiable reward can be given at some point of the process (can be as simple as success = 1, failure = 0). With some environments this requires creativity on the programmers part to define the environment in a finite number of ways. As an option, the learning process can be online or offline. What this means is that the learning can take place at a certain time and then use a greedy algorithm once a function evaluation has been trained or that the program can be learning as it is performing the actions.

Temporal DIfference Learning Theory

There are many different equations that can describe a method to Reinforcement Learning. The most basic understanding of Reinforcement Learning can be understood through the Temporal DIfference learning method, which was one of the first to be created. Scholarpedia.com offers a great description of this method:

We consider a sequence of states followed by rewards: . The complete return to be expected in the future from state is, thus: , where is a discount factor (distant rewards are less important). Reinforcement Learning assumes that the value of a state is directly equivalent to the expected return: , where is here an unspecified action policy. Thus, the value of state can be iteratively updated with: , where is a step-size (often =1).

Though this equation seems complex, it can be explained rather simply. Since states and rewards can be thought of as discrete events states, can be assigned rewards. The reward of a state is the value of expected reward of the next state with a discount factor to make sure that distant rewards are not as important as immediate rewards. The algorithm then evaluates the value of each state in a given environment and determines the value of the state based on the current policy being used. This state can then be updated as the sequence continues by adding the value of the current state to some fraction of the difference between the reward actually received and the supposed value. The most important part of this algorithm is that policy will eventually converge to the optimal policy. Another very important part of the Temporal Difference method are eligibility traces. In chess usually there will be one or two moves in the game that determines the outcome of the game. A simple mistake or an ingenious move can be the key to winning or losing. For the temporal difference method, part of the solution is finding out which moves specifically caused the change of the expected outcome of the game.

There are many other forms of reinforcement learning such as Q-Learning and SARSA, but these are just specific algorithms for specific situations. Essentially they are all the same in nature, but the algorithms work differently to emphasize certain parts of the algorithm. Some techniques include limiting the depth of the search for the expected reward, using a fixed policy for the learning algorithm rather than an undetermined policy, or SARSA learning which learns while it is performing actions.

Temporal DIfference Learning Psuedocode

To better understand the temporal difference algorithm it is often best to use psuedocode. The following example demonstrates the temporal difference method of learning. (Stuart and Russell, p. 769)

We consider a sequence of states followed by rewards: $s_t, r_{t+1}, s_{t+1}, r_{t+2}, \ldots, r_T, s_T$. *The complete return* R_t *to be expected in the future from state* s_t *is, thus:* $R_t = r_{t+1} + \gamma^1 r_{t+2} + \ldots + \gamma^{T-t-1} r_T$, *where* $\gamma < 1$ *is a discount factor (distant rewards are less important). Reinforcement Learning assumes that the value of a state* $V(s)$ *is directly equivalent to the expected return:* $V(s) = E_\pi(R_t | s_t = s)$, *where* π *is here an unspecified action policy. Thus, the value of state* s_t *can be iteratively updated with:* $V(s_t) \rightarrow V(s_t) + \alpha[R_t - V(s_t)]$, *where* α *is a step-size (often =1).*

What is happening is that using an estimate of the expected reward for all of the states possible given the current one, the value of the previous state is updated its value. Initially the reward is completely unknown and is just a guess, but over time these guesses converge towards the correct values. The alpha and gamma values can be changed for each problem and tend to have different values depending on the problem. The Temporal DIfference method can also look any given number of plys deep or move ahead. This simplest example is TD(1) where it searches 1 ply deep, or TD(2) which searches 2 plys deep, but could be set to any number.

Why Temporal DIfference Learning Works

The Temporal DIfference Learning algorithm does not tell the entire story of how it learns. Part of the problem with greedy algorithms, algorithms that always attempt to perform the best move based on a fixed policy, is that once the algorithm finds an optimal solution they never look back to see if another solution could have been better. To compensate for this an approach using Temporal DIfference learning is using guesses and a method called e-greedy to determine which move to make next. Often it is not the best approach to always take the best current perceived move at the time. Using e-greedy solves this problem. A boundary called e is set to be the maximum amount of exploration that can be done. This e often has to be tuned to fit the specific problem. A problem with a small state space will have a very small e, but with a large state space will have a large e. Thus when a program is trying to decide which move to take next, once in a while it will take an exploratory move instead of the first choice. Since the states visited have often not been evaluated guesses are given to their values. From these guesses more guesses are made about the state. After a number of trials these guesses are tuned more towards their actual values and go towards the optimal state.

Variations and Range of Reinforcement Learning Problems

Reinforcement Learning can be abstracted to cover a wide range of problems and techniques already used. In the Temporal DIfference method α helps define the amount of knowledge learned. If it is set to 0, then this becomes a greedy algorithm and can be defined as the expectiminimax algorithm which is the traditional algorithm. If it is set to 1, the policy is set to anything, and the search is a full depth search, then this becomes the Monte Carlo method.

Approximation Functions

One of the key parts to the Reinforcement Learning problem is evaluating a state. In Reinforcement Learning developing approximate value of a state is a vital part of the method. There are many ways to do this. A table with a list of all the possible states can be made and be given a score. This requires a finite amount of states and preferably a small number since two similar states will not share any information. Another method would be to use an Artificial Neural Network. By constantly backing up the values of the ANN,

a function approximation can be made. The ANN has the advantage of being able to cover a large state space. If the state space, for example, is a chessboard forcing the program to learn every different state would be impossible, but with a neural network it is able to handle these situations.

Approximation functions are the key to solving large state problems. "However, using function approximators requires making crucial representational decisions (e.g. the number of hidden units and initial weights of a neural network). Poor design choices can result in estimates that diverge from the optimal value function (Baird, 1995) and agents that perform poorly. Even for reinforcement learning algorithms with guaranteed convergence (Baird and Moore, 1999; Lagoudakis and Parr, 2003), achieving high performance in practice requires finding an appropriate representation for the function approximator". (Whiteson, 2006, p. 878)

Scope of Project

The challenges and possibilities of making the best Connect Four program are ultimately what the project is about, however, in order to research the problem I will test my program in multiple fashions to see if there are any insights that would be useful for all AI. My project consists of developing a Connect Four program using the temporal difference learning method and ANNs. Once developed, the program will be tested with multiple parameters to determine if it can be improved. This project will allow me to look at the viability of reinforcement learning when compared to deep search algorithms in Connect Four, which can lead to usefulness in other games.

My research incorporates an AI program for Connect Four using the Mini-max method and comparing it to the Reinforcement Learning algorithm. I tested many different factors in order to determine the optimal Neural Network size, the learning rate for the algorithm, and finally the difference of hardware and memory that will affect the performance of the Mini-max algorithm. The Reinforcement Learning Program will first begin by training itself how to play. After a set number of games it has trained, it will then begin to play against the Mini-max program to see how it does. Data will be recorded and training will continue. These steps will take place until no improvement is seen in the increase of the number of games played. Likely there will be a breaking point at which the Reinforcement Learning Program will vastly improve against the Mini-max program. Arthur Samuel used reinforcement learning in backgammon and were able to train it to defeat even the best masters (Sutton & Barto pg. 267). The hope is that it also applies to games that are more topological or positional in nature like Connect Four, Chess, and Go.

Results

After conducting research, even after trying multiple parameters, my results were negative. The Reinforcement Learning program simply did not perform well enough. My program was tested against my earlier program using a variation of 4 different parameters and won few games except for the occasional fluke derived from a flaw in the first program.

Analysis

My analysis of the program emphasizes that the reinforcement learning, when applied to positional games, does not take advantage of the dynamic situations that are present. It is very rare that two games are played exactly alike, and thus the reinforcement learning algorithm does not have any time to adapt to new positions and is not capable of learning positional moves. Further study and thought makes me believe that a dynamic version of reinforcement learning, like Monte Carlo, will succeed on the basis of its ability to learn each position dynamically instead of relying on

memory. In conclusion, although my program was not a success, my project demonstrates that, even in a simpler game like Connect Four, reinforcement learning does not appear to be the best method for play.

BIBLIOGRAPHY

Alpaydin, Ethem. Introduction to machine learning. Cambridge, Mass: MIT P, 2004.

Russell, Stuart J., and Norvig, Peter. Artificial intelligence a modern approach. Upper Saddle River, N.J: Prentice Hall/Pearson Education, 2003.

Sutton, Richard S., and Andrew Barto. Reinforcement learning an introduction. Cambridge, Mass: MIT Press, 1998.

Tesauro, G. 1995. Temporal difference learning and TD-Gammon. Commun. ACM 38, 3 (Mar. 1995), 58-68. DOI= http://doi.acm.org.ezproxy.lib.csustan.edu:2048/10.1145/203330.203343

Whiteson, S. and Stone, P. 2006. Evolutionary Function Approximation for Reinforcement Learning. J. Mach. Learn. Res. 7 (Dec. 2006), 877-917.