CS229 Final Report

The Applicability of Machine Learning Concepts to Game Artificial Intelligence

Garrett Gutierrez
garrettg@stanford.edu

Qiujiang Jin
qiujiang@stanford.edu

December 16, 2016

Abstract

The purpose of this study is to examine the applicability of machine learning concepts to general game playing. Results were compared against a Monte Carlo Tree Search player using a propositional network to represent game states. Linear regression proved to be effective at determining which state propositions, if any, contributed directly towards higher utility for the player, and in games where such propositions did exist the hypothesis function derived by the linear regression method was effective. Conversely, the hypothesis function performed especially poorly in the cases where what constituted a more advantageous state was more complex, particularly in cases where the goal was more directly correlated with the number of turns played in the game than with specific propositions more directly in the player's control.

1. Introduction

While there are many robust artificial intelligence algorithms that are capable of general game playing, they are often limited with regards to how well they trade accuracy for efficiency. In a time when real-time game applications with complex game states require AI to make believable actions within fractions of a second, trade-offs between efficiency and accuracy are inevitable and are thus a constant consideration for those implementing game AI. We aim in this paper to examine the applicability of machine learning concepts within the context of general game playing.

2. General Game Playing

The purpose of this paper is to measure to efficacy of machine learning concepts within the context of general game playing algorithms. In this paper, a general game playing algorithm (also referred to as a general game player) is any algorithm capable of playing games with no prior knowledge of the their rules or structure. During runtime, the general game player is passed a description of the game (from which it can determine states, legal actions, goal values, etc.), a start clock time, and a turn clock time. The player has the amount of time specified by the start clock time to perform any preprocessing necessary. Then, every turn, it has the amount of time specified by the turn clock time to decide which action to play. If time runs out during a turn, a random action is chosen for the player. For games with multiple players, all actions are chosen simultaneously. In games where players take turns, the player whose turn it is not will always have one action available to them, which is to do nothing [2].

3. Data and Features

There is no precompiled game state data because, by the rules of GGP, the player is permitted no knowledge of the game before the game begins. Feature vectors for game states are n length vectors where n is the number of propositions in the game. For example, if proposition j in checkers is "the red player has captured 4 total black pieces", then $x_j^{(i)} = 1$ if the red player has captured 4 total black pieces in state i. Otherwise $x_j^{(i)} = 0$.

4. First Model: Post-Game Linear Regression

We began measuring the efficacy of machine learning in the context of general game playing by determining a rough upper limit for how effective a linear regression algorithm implementing batch gradient descent could be expected to perform in the case where the time to learn on the game data was essentially unbounded.

4.1. Measuring Efficacy

To measure how effective the post game linear regression model was we chose to compare it against a standard Monte Carlo Tree Search player implemented during the game itself. A Monte Carlo Tree Search player builds a tree where each node represents a game state and the children of a node represent all possible states reachable from the parent state for each joint combination of actions taken by all players [1]. Within the start and turn time the player explores as much of the tree as possible and assesses the predicted value of each node by choosing a random path downward from the node to a terminal state. The average value of the paths down from a node represents the node's value, and at the end of a turn, before the clock runs out, the player picks the action which leads to the nodes with the best average values. We chose to compare the linear regression model trained after the game to the Monte Carlo Tree Search player implemented during the game because it was the Monte Carlo Tree Search data that the linear regression model would be trained on, so while Monte Carlo Tree Search players are by no means perfect, we judged it unreasonable that a linear regression model trained on state utility data derived by a Monte Carlo Tree Search player would perform better than said player.

This Monte Carlo Tree Search algorithm uses as its internal state machine a propositional network with backwards propagation. This algorithm is effective on games where positive-valued goal states are not especially sparse. Our linear regression model will not be compared against Monte Carlo Tree Search in a case where Monte Carlo Tree Search itself performs poorly on the game being played.

4.2. Linear Regression: Batch Gradient Descent

The post-game linear regression model was implemented using batch gradient descent on state utility data derived by a Monte Carlo Tree Search player after the game was completed. This was not been tested against Monte Carlo Tree Search during game play. The learning algorithm was trained on the game states explored by the Monte Carlo Tree Search algorithm and as the target y values the approximated utilities derived by the Monte Carlo Tree Search player were used. The amount of time given to the learning algorithm was unbounded. This allowed us to derive an upper-bound on the efficacy of linear regression.

It should be noted that our step size ranged from $\alpha = 0.01$ to $\alpha = 0.0001$ and the number of iterations of repeated batch gradient descent ranged from 100 to 10000, depending on what completed with reasonable amount of time (within 10 minutes).

After learning was performed, the accuracy of the learned heuristic function was measured by again selecting random paths downward from the root node to the terminal node. For each non-terminal node along the path, the ideal child state determined by Monte Carlo Tree Search was compared against the ideal child state determined by the learned hypothesis function h_θ. If these algorithms made the same decision regarding the ideal child state, this experiment was considered a success. The total amount of successes was divided by the total number of checks made to determine the efficacy of the hypothesis function. This decision was made because in practice the actual predicted utility of a state is irrelevant; what matters is how it compares against its neighbors.

4.3. Results

The algorithm was tested on a variety of relatively simple single player and multiplayer games. The approximate degree (number of legal actions per turn) and height (number of turns in a game) are shown in Table 1 to give some indication on the size of the game tree.

Game	Degree	Depth	Accuracy
3-puzzle	2	6	0.61
Buttons and Lights	3	6	0.21
Hunter	2-4	14	0.08
Knight's Tour	2-6	27	0.03
Connect Four	8	14	0.23
Checkers	9-10	49	0.71
Alquerque	10	29	0.72

Table 1: *Results for each game tested. Both degree and depth describe the game tree (where each node is a state) are approximations.*

4.4. Interpretation of Results

Linear regression using batch gradient descent appeared to do much better in games where the preferability of a state was immediately apparent. For example: in both Alquerque and Checkers, a player's goal is measured by how many of the enemy pieces they capture. Every number of captured enemy pieces corresponds to a state proposition which itself corresponds to both a feature in the feature vector and a goal value. Because there is a direct correspondence between these features and the utility of a state, good assessments can be made regarding which states are preferable based on the value of these features within a state's feature vector, leading to a stronger heuristic function.

In comparison, with games like Knight's Tour and Hunter, the goal value is directly and most strongly associated with how many turns have been taken. In Knight's Tour specifically, the goal is to make the game go on for as long as possible, and how long a game could potentially go on is something that is not immediately apparent within a state's feature vector at a level reducible to which individual propositions are true independent of the others. This means that the difference between the utilities of a preferable and of a non-preferable state are often orders of magnitude smaller than the utilities themselves, it is difficult for a linear regression algorithm to capture what makes one state preferable compared to another.

The average breadth and average depth of the game tree did not appear to have as much of a significant bearing on the efficacy of the learned hypothesis function.

5. Second Model: Pre-game Linear Regression

Since linear regression seemed to be accurate on games with advantageous game propositions, we realized that we could use linear regression to create a greedy player that prefers states that have as true propositions that are recognized to be advantageous. Such a player could be effective in situations where the start clock gave the player enough time to compile sufficient training data but the play clock was minimal enough that a regular Monte Carlo Tree Search player would essentially be forced to return a random action. Algorithm 1 shows the adjusted back propagation function used by the linear regression player during the start clock. During the play clock, it merely judges the best action based on which resulting states maximize $h_\theta(s)$. We chose to incorporate linear regression into Monte Carlo Tree Search because the player, by the rules of General Game Playing, does not know how much time it will have during the playing phase, and so will not know for certain what model it will have to rely on.

```
Data: Terminal state $s$ and associated utility $u$
      derived by single depth charge of MCTS. $\theta$
      derived from previous backward
      propagations and step size $\alpha$.
Result: Updated Monte-Caro game tree and $\theta$.
while $s \neq null$ do
    $s.utility \leftarrow s.utility + u$;
    if $s.recorded == false$ then
        for $j \leftarrow 0$ to $n$ do
            $\theta_j \leftarrow$
            $\theta_j + \alpha(u^2 - h_\theta(s.features))s.features_j$;
        end
        $s.recorded \leftarrow true$;
    end
    $s \leftarrow s.parent$;
end
```

Algorithm 1: Altered backwards propagation algorithm incorporating linear regression.

6. Results

The player that conducts pre-game linear regression, in its current state, does not appear to perform any better than a random player during games such as Checkers and Alquerque. There is a significant degree of difficulty involved in choosing an appropriate step size. In addition, training θ on the initial judgment of the utility of a state means we are training it on the weakest possible meaningful assessment of a state's utility. We tried an alternative model that simply ignored if a state has been recorded or not and trained on it anyways, but we still did not achieve results that were notably better than a random player.

7. Conclusions

While establishing an upper bound on the efficacy of linear regression proved promising, more work could, in the future, potentially go into establishing how to measure the amount of training necessary to reach that level of efficacy. However, it may also be possible that a greedy algorithm that involves directly examining the propositional network to determine propositions associated with specific goal values may have superior performance in certain circumstances in addition to being easier to implement. Such an algorithm would not need to perform training to establish what propositions are preferable, but would instead rely directly on the logic of the game itself. There is the potential for situations where the relationships between individual propositions and specific goal values are, however, more nuanced and a model of logically assessing the value of these propositions might be insufficient. Judging the utility of such a method in comparison to the linear regression model described would require exposure to a wider variety of single player and multiplayer games.

8. References

[1] C. Browne et al, "A Survey of Monte Carlo Tree Search Methods" in *IEEE Transaction on Computational Intelligence and AI in Games*, vol. 4, no. 1, March 2012.

[2] M. Genesereth et al, "General Game Playing: Overview of the AAAI Competition" in *AI Magazine*, vol. 26, no. 2, Summer 2005.