Imitating Play from Game Trajectories: Temporal Difference Learning versus Preference Learning

Thomas Philip Runarsson, Senior Member, IEEE, and Simon M. Lucas, Senior Member, IEEE,

Abstract—This work compares the learning of linear evaluation functions using preference learning versus least squares temporal difference learning, LSTD(λ), from samples of game trajectories. The game trajectories are taken from human competitions held by the French Othello Federation1. The raw board positions are used to create a linear evaluation function to illustrate the key difference between the two learning approaches. The results show that the policies learned, using exactly the same game trajectories, can be quite different. For the simple set of features used, preference learning produces policies that better capture the behaviour of expert players, and also lead to higher levels of play when compared to LSTD(λ).

I. INTRODUCTION

For board games an evaluation function is commonly used to rate how favourable a board position is for a given player. This evaluation function can then be used within a game-tree search algorithm to provide an effective playing policy. Traditionally this approach has been most commonly used in conjunction with minimax game-tree search. Computing the evaluation function of a policy is usually performed by minimizing Bellman-residuals [1]. The most effective approaches employ least squares temporal difference learning, LSTD(λ) [2], [3]. This algorithm has no control parameters and therefore eliminates the problem of parameter choice leading to poor performance. When using LSTD the evaluation function will have a linear form in feature space, though the features may be non-linear functions of the board state. Within the constraints of learning a linear function, LSTD aims to approximate the expected future payoff. The game playing policy is then indirectly represented as greedy moves with respect to this function. However, policies may be easier to represent than value functions [4]. This has motivated a number of researchers to model the policy directly as a classification problem [5], [6], [4]. These methods use rollouts to estimate the value of alternative moves at a given board position. Then, if a move has a statistically greater value than all other moves it is added to a training set with a positive label, while the rest are also added to the training set with a negative label [5]. Recently it has been argued in [7] that minimizing Bellman-residuals is unnecessarily complex, since predicting precise future payoffs is not necessary for making optimal moves. Furthermore, they argue that for the latter approach the prediction of single moves neither suggests alternative actions nor offers any means for proper exploration [7]. Creating positive and negative move examples is different to preference based learning and will result in a different evaluation function, even when the same linear architecture is chosen.

Here we will investigate learning an evaluation function for Othello, which was held as an IEEE CEC competition2 for three years. Othello is interesting for several reasons, including the way that game states are highly volatile, and that piece difference during the middle of the game is very deceptive, with stronger positions often showing poor piece difference. The strongest Othello program is Logistello3. The evaluation function used by Logistello also has an essentially linear architecture (but with a sigmoid squashing function at the output) based on 1.5 million pattern-based features using different evaluation functions at 13 different game stages, \( g_s = \max \{0, \lfloor(|\text{discs} - 13)/4 \rfloor\} \). The training data used by Logistello is based on some 80,000 games generated by the program against another Othello program (Kitty). Towards the end of the game the positions are labelled perfectly since an endgame negamax search is used. Values of middle and opening game positions are approximated based on the game outcomes that followed those positions. Logistello then uses a gradient descent algorithm to estimate the model’s parameters. Logistello’s approach [8] corresponds more closely to the supervised learning approach, or LSTD(1), using linear regression to learn the value of positions labelled with the final disc differential estimate. This approach yields significantly better performance than Buro’s previous work [9] where the positions were labelled by the probability of winning. Clearly, labelling on the probability of winning or the outcome of the game is a more general approach, and valid for all board games.

When observing game trajectories created by human game playing it is difficult to automatically label any particular move as negative. The game logs only give us information on moves selected, i.e. the policy. Even if the outcome of a game is a loss it does not necessarily mean that all the moves made during game play were poor. Clearly, imitating human-like play is not necessarily optimal, though it may create players that are more interesting to play against. This would be especially true if it proved possible to imitate particular famous players rather than human-like play in general. To learn to play from game trajectories, the most applicable approaches are those of minimizing Bellman’s residual error, using for example LSTD(λ), and preference learning. However, what these methods learn can be quite different and it is this difference which we clearly demonstrate in this

1http://www.fiothello.org/

2http://algoval.essex.ac.uk/othello/html/Othello .html
3http://skatgame.net/mburo/log.html
paper.

Preference learning is used as an alternative means of estimating a value function, equivalently to that presented by the first author in [10], where the method was applied to learning heuristic evaluation functions for combinatorial optimization problems from known optimal solution trajectories. This idea also has similarities to the work on maximizing concordance [11]. The state of the game is analysed at each game stage, similar to [8], in order to make best use of the available information, with the focus on making correct decisions rather than minimising the mean square error. This shift in focus is important. The key insight is that making the correct choice is what really matters, and interestingly, this may make the problem easier to learn. Results in this paper support this view.

The rest of this paper is structured as follows. The next section discusses the Othello game trajectories used in this study, followed by a brief description of the LSTD(λ) and preference learning algorithms employed. The results of the comparison are presented in section III and section IV concludes.

II. Othello Game Trajectories

Othello, like many boardgames, fits the model of a two-player, turn taking, zero-sum game where the utility values at the end of the game are equal and opposite. In this paper we use a weighted piece counter as the form of value function to be learned by each approach. The 8×8 board is unwound as a 64 element vector φ. Each element is 0 for an empty square, +1 for a black counter and −1 for a white counter. Hence, black is the maximising player and white is the minimising player. The weighted piece counter has a vector w of 64 weights, one for each square on the board. The value of a board is then calculated as the scalar product wφ. In general φ could be any vector of features, but here the investigation is limited to using the board pieces directly. When choosing a move a one-step-lookahead is performed. The resulting board (after- or post-decision-state) is evaluated and the move with the best corresponding evaluation is chosen.

Othello game logs from human players in the French Othello League were used to create game trajectories.

A. Least Squares Temporal Difference Learning

The essence of temporal difference learning (TDL) is to learn that states that are close in game trajectories should have similar values. In traditional TDL a state’s value is updated on-line as a game is played. After the update is calculated it is reduced by a factor α (the learning rate) before being used to update the state. If α is too high then learning can be unstable. If α is too low then learning can be too slow. Tuning α to achieve acceptable performance is a significant problem in TDL. Hence in recent years there has been interest in more sophisticated TDL algorithms that do not require a step size to be set. This is the approach taken by Least Squares TDL, LSTD(λ), and results for this algorithm on toy problems such as the Boyan Chain are indeed impressive [3]. Our implementation is based directly on [3], and we verified its operation on the Boyan Chain test problem. The implementation of LSTD(λ), for two player games, is presented by algorithm 1

<table>
<thead>
<tr>
<th>Algorithm 1: LSTD(λ). Matrix A has dimension n × n and φ₀, φ', p, b, w are vectors of dimension n × 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>input</strong></td>
</tr>
<tr>
<td><strong>output</strong></td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
<tr>
<td>13</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>15</td>
</tr>
<tr>
<td>16</td>
</tr>
</tbody>
</table>

B. Preference learning

The aim of preference learning is to make the correct choices rather than minimise some surrogate of such as the mean square error. We cast preference learning in the context of board game playing by using an evaluation function in conjunction with one-ply lookahead. Each move is made by considering all the after-states reachable by the set of legal single moves from the current board. Hence, a training set can be constructed from a set of game logs (trajectories) as follows. For each non-terminal board state, the set of next possible board states is constructed. The one chosen in the current game log is labelled as the correct move, and all others are labelled as incorrect. This is presented to the preference learner as a set of constraints: the constraint to be making the correct decision with a clear margin, arbitrarily chosen to be 1.0. In other words, the learner aims to satisfy this constraint for the maximising player:

\[
[w(\phi_j − \phi_k)] > 1 \forall j \in C^+, k \in I^+ \quad (1)
\]

and similarly for the minimising player:
where \( C \) and \( I \) are the correct and incorrect board states respectively, with the superscript indicating whether the player is maximising or minimising. We used the LibLinear\(^4\) machine learning package to find \( w \) using its default settings which are: L2-regularized L2-loss support vector classification, parameter \( C = 1 \), and no bias term.

### III. RESULTS

Results are reported in three ways: how well the algorithms were able to match the human decisions made, how similar the learned weights are to the standard hand-crafted heuristic weights, and the level of play achieved by each learned weight vector against each other one when used in a one-ply minimax search. Figure 1 shows how the decision accuracy varies with the stage of the game for each approach. The top set of lines indicates the percentage of correct pair-wise decisions while the bottom set indicates the percentage of correct move choices. Table I shows the underlying numbers, with the bottom row showing the mean branching factor, the total number of patterns, and the mean accuracy for each approach. Note how preference learning significantly outperforms LSTD until the final stages of the game, when LSTD performs slightly better. By tuning the parameter \( \lambda \) we find that LSTD(0.9) outperforms LSTD(0), the exception being at the end of the game.

We analysed the weight vectors learned in each case using graphical plots, as seen in figure 2. One of the most important things to learn in Othello is the value of playing in the corner, and the danger of playing next to the corner. Only preference learning was able to learn both these things. Both LSTD variants learned the value of playing in the corners, but were unable to learn the danger of playing adjacent to a corner, though at least LSTD(0.9) rated this as neutral, whereas LSTD(0) rated this as being a good move. This observation coincides with the way that LSTD learns better in the later stages of the game, at which stage the adjacent cells to a corner may well have been flipped to the colour of the corner occupier, and hence they would no longer show up as being poor moves.

Playing strength was estimated from a full round robin league where each weight vector was used to play each other one using one-ply minimax search from the same 1000 randomly chosen positions. We then used BayesElo\(^5\) to rank the players and to assess the likelihood of superiority. The calculated order was HEUR > PREF > LSTD(0.9) >

- \( \mu (\underline{d}) \) 8.6 (437883) 13.8 15.6 18.1 28.2

---

4\[http://www.csie.ntu.edu.tw/~cjlin/liblinear/\]

5\[http://remi.coulom.free.fr/Bayesian-Elo/\]
LSTD(0), with each higher rated player having a likelihood of superiority of 0.99 or greater than every player below it.

IV. CONCLUSION

LSTD is a powerful temporal difference learning algorithm with attractive convergence properties. However, convergence to the Bellman residual error is not actually what is required when learning to imitate a set of human players, or when aiming to achieve high performance game play. What matters is learning to make the correct decisions, and preference learning is explicitly formulated to do this.

This paper presented results of using LSTD and preference learning to learn to play Othello from logs of games from human players in the French Othello League. In each case learning was used to estimate the weights of a weighted piece counter. Preference learning outperformed LSTD in three ways: it learned to better match expert human decision making (apart from at the end of the game), it produced weights that were more similar to the standard heuristic weights, and it produced better performing players.

The results have important implications for learning in games. Preference learning has a great deal to offer in better imitating human styles of play to provide more satisfying opponents for human players. Ongoing work is investigating whether similar results are obtained when using high-performance value functions based on n-tuple features [12].

REFERENCES