Learning a Reversi Board Evaluator with Minimax

Kevin T. Engel
University of Maryland, College Park, MD 20782

A board position evaluator is a crucial component for strong computer play in many games such as checkers, chess, and Reversi. The board evaluator is typically trained using pre-existing game data, an approach which is generally non-optimal, especially in the early stages of a game. Instead, we propose a new method which relies on Minimax search to train a series of models backwards from the endgame, propagating information about endgame scoring backwards to earlier positions. In the limit of perfect models, our method is optimal, converging to the full Minimax board evaluation. We test our method experimentally by training a simple Reversi model using both our Minimax method and high-level tournament game data. When played against each other, our model outperforms the game data model, averaging 5 more stones per game, and winning approximately 60% of the matches.

I. INTRODUCTION

The Minimax algorithm has become a standard feature of computer game players. Minimax is used to identify the best moves in a game tree generated by each player's legal actions. Terminal nodes represent finished games; these are scored according to the game rules. Scores can then be propagated backwards through the game tree by assuming that each player plays perfectly, always choosing the action which maximizes their eventual endgame score. For short games this algorithm can be used exactly, essentially solving the game. For longer games however, the full search is infeasible, and the tree search is usually halted at some fixed depth. Because these no longer correspond to finished games, a method for scoring intermediate states is required. For board games like checkers, chess, and Reversi, the number of intermediate states is immense, one for each unique board state, and a model for evaluating board positions is required.

In this work our focus is not on the models, but on the methods used to train the models. The simplest and most commonly used method is to train a board evaluator on previous game data. For this training data, a board's score is given by the game's outcome. Although this produces reasonable results, there is no guarantee that the model is learning the optimal strategy, especially for early boards where the connection between the current position and end game score can be quite tenuous. Instead, we propose a new method which leverages the perfect play of Minimax in order to propagate endgame information backwards to earlier game stages. In our method, models are trained in stages, starting near the end game. Training data is provided by generating random boards which are scored using Minimax and the previous model. We give more details about our method in Section IV, but first we present some background information and related work in Sections II and III. In Section V we present a simple Reversi model and discuss the various options and parameters which should be explored to give the best results. These are experimentally determined in Section VI which culminates in a series of Reversi matches between a Minimax trained model and a game data trained model.

II. BACKGROUND

Reversi is a two player game in which opponents take turns placing stones of their color (white/black) onto the board, traditionally an 8×8 square grid. A legal move consists of a stone placement to an empty square which traps a continuous sequence (either horizonally, vertically, or diagonally) of your opponent's pieces between two stones of your color. All pieces which have become trapped in this manner are then flipped to the opposing color. If a player has no legal moves, they must pass, allowing their opponent to play again. The game begins with a board configuration consisting of 4 stones, 2 of each color placed diagonally, forming a square at the center of the board. Players then alternate turns until the board is full or neither player has a legal move. At this point, the player with the most stones of their color on the board is the winner.

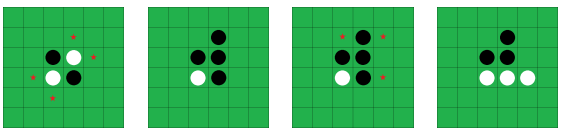

FIG. 1: An example of beginning Reversi gameplay. Red stars indicate legal moves for the active player.

Developing a computer Reversi player requires a method for identifying the best move from any board position. Given unlimited computational resources, the ideal approach would be to exhaustively search the entire game tree, assuming both players play perfectly. The leaf nodes of the game tree represent endgame boards and can be scored for the white player as *white stones* − *black stones*. Non-terminal nodes represent boards where the active player has one or more legal moves. The assumption of perfect play implies that these nodes can be scored as the maximum (minimum) score of their child nodes for white (black) active player. By propagating the scores back to the starting position, the computer player can then choose the move which gives the best score. This is known as the Minimax algorithm

and it has become a standard approach to game AI.

For short games such as tic-tac-toe, the Minimax algorithm can be used exactly as described, but for longer games like Reversi, chess, Go, etc, such a calculation is only feasible for board positions near the endgame. Because the computational cost is exponential in the search depth, a full search is impractical for most board positions. The runtime of the Minimax algorithm can be improved using alpha-beta pruning [1], but the overall exponential dependence on the search depth cannot be avoided. Because of this, Minimax is usually run with a fixed search depth, resulting in leaf nodes which are not endgame boards. The exact score for these boards is therefore unknown, and a board position evaluator must be used to predict the score. Much of the work in game AI centers on creating a good position evaluator - finding important features and designing or training a model to predict the quality of intermediate board positions.

III. PREVIOUS WORK

One of the first world-class Reversi programs was created by Paul Rosenbloom in 1982 [2]. *Iago* used a linear combination of mobility-based features as well as edge feature weights which were drawn from a precomputed table. The table holds values for each of the 6561 (3^8) possible edges, and is computed using an iterative algorithm. Each edge is assigned an initial value given by the sum of hand generated weights for each square which attempt to measure the importance of position and stability. The edge values are then updated by allowing for legal moves which can turn one edge configuration into another. If the transformation results in a higher edge weight, the original edge weight is replaced by the higher one. This process is repeated for several iterations until the edge weights have converged. Instead of training a board evaluator, *Iago* designed one - all of the weights were chosen by hand. A limited set of weight variations were considered, and the best set was chosen by playing the different versions of *Iago* against themselves.

A similar program known as *Bill* was designed in 1986 by Lee and Mahajan [3]. The basic structure was the same, including the table-based evaluation scheme, but *Bill* introduced efficiencies in evaluating board positions and game tree search which gave it roughly a 2-ply advantage over *Iago*. The edge weights were initialized as before, but a more complicated scheme involving Minimax search and fabricated probabilities was used to generate the final edge weight table.

In 1988, *Bill* was substantially improved by replacing some of the old hand-tuned weights with ones which were trained using game data [4]. 3000 games were generated by self-play using the old version of *Bill*. Boards were then labeled as win/loss depending on the game outcome. These examples are used to learn the parameters of a two class multivariate normal distribution in the relevant board features ($\mathbf{x}$):

$$p(\mathbf{x}|c) = \frac{1}{2\pi^{-N/2}\left|\Sigma_c\right|^{1/2}} e^{-\frac{1}{2}(\mathbf{x}-\mu_c)^T\Sigma_C^{-1}(\mathbf{x}-\mu_c)}. \quad (1)$$

New boards can then be evaluated using the scoring function:

$$g(\mathbf{x}) = \log(p(\mathbf{x}|\text{win})) - \log(p(\mathbf{x}|\text{loss})). \quad (2)$$

From evenly matched starting positions, this new version of *Bill* defeated the old one by a ratio of 2:1.

The final Reversi computer player which we will consider is *Logistello*, an overview of which can be found in [5]. This program incorporated several advances over its predecessors. Search is an important part of game AI, and Logistello was able to search deeper than its opponents by using ProbCut - an algorithm which prunes subtrees which are probablistically irrelevant [6]. *Logistello*'s creator, Michael Buro, also demonstrated that using logistic regression for model training achieved better results than the quadratic disciminator used by *Bill* [7]. *Logistello* finally eliminated hand tuned weights, as all weights were trained using game data generated by about 60000 matches between other expert level Reversi computer players. As before, the boards were initially scored using the game outcome. Given the large numbers of games, though, some boards may have multiple successor boards present in the dataset; in that case their scores are updated using the Minimax principle, giving a more accurately scored training set.

Reversi AIs have evolved from using hand tuned weights to learning weights with expert-level game data. This has become the dominant approach to creating a board position evaluator. An alternative method is the use of self-play and reinforcement learning, which has achieved some success in the past, most notably in backgammon [8]. Reinforcement learning techniques have been employed in many games, for a general review see [9]. For Reversi in particular, a number of attempts have been made [10–12] with varying levels of success.

IV. LEARNING A BOARD EVALUATOR WITH MINIMAX

As discussed in the previous section, past attempts at creating a strong Reversi (and other games) AI have approached the problem of board evaluation primarily as a modeling challenge - identifying the useful features and selecting the correct model. While this is undoubtedly important, little attention has been paid to the data used to train the models. The usual approach is to use prior existing game data where the match outcome determines the score of all board positions encountered in the game. If both players played perfectly, this would be the correct approach - the score would be the actual Minimax value and the model would learn to predict these values. Both human and computer players are not perfect, however, and while the occasional mistake may act as noise

to be overcome by increased sampling, any collective flaw in board evaluation by the players will be reproduced in models which train on this data.

We propose an alternative method of model training which uses the Minimax algorithm to propagate board evaluations backwards from the endgame. We describe here its application in Reversi, but it should be adaptable to other games. It is a simple idea - we first generate a set of boards with N stones by making moves randomly from the starting configuration. These boards are then scored using the Minimax algorithm (N should be chosen sufficiently close to the endgame to make this practical) and an N stone model is trained. Next we generate a set of boards with $N - M$ stones by again making random moves. Scores for these boards are generated using Minimax with a fixed depth of M, with leaf nodes scored using the N stone model. In this way we generate models for N, $N - M$, $N - 2M$, ... stone boards.

In the limit of perfect models (i.e sufficiently complex to perfectly capture the board evaluation) then this method is clearly sound; given enough data, each model would learn to predict the Minimax value exactly. This is not the case when training on game data, any common mistakes made by the players will also be learned by the model. A different, less-used approach to model training involves self-play and reinforcement learning. Like our method, this one does not rely on potentially flawed game data, however there are issues with fully exploring the search space and converging to non-optimal minima[13]. Our Minimax method does not suffer from these drawbacks; given a reasonably good model, we expect it will outperform these other training methods.

V. EXPERIMENTAL SETUP

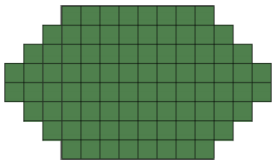

FIG. 2: 88 stone Reversi board

We explore various aspects of our Minimax training method using the non-standard Reversi board shown in Fig 2. The larger board helps ensure that our method works for longer games, as well as demonstrating the value of our method when dealing with games for which no game data is available. For this board we decided to use the following set of simple features:

- **Parity** - +1 if black/white is the active player and the number of stones on the board is even/odd, -1 otherwise.
- **Mobility** - number of distinct legal moves a player can make.
- **Frontier** - number of empty spaces adjacent to a player's stones.
- **Flippable Pieces** - sum of the number of flipped stones resulting from each legal move of a player's opponent (some overcounting may occur).
- **Board Features** - player's occupancy of unique board spaces. The 88 stone reversi board has 2 mirror symmetries, and therefore only 22 board features.

For all features except parity, the features are first calculated for the white player, then for the black player, and the difference is reported as the result. As an example, if the white player has stones on 2 of the corner squares, the black player has a stone on 1 of the corner squares, and the last corner is empty, then the board feature corresponding to the corner will have value $2-1=1$.

We consider three different types of models which utilize these $n_f = 26$ features. When training on m boards, we will have an $m \times (n_f + 1)$ feature matrix D (a column of ones is added to include an offset) and score vector $\mathbf{y}$. Our first model is a linear one, minimizing the objective function:

$$f(\mathbf{w}) = \frac{1}{2} \left(D\mathbf{w} - \mathbf{y} \right)^2 . \tag{3}$$

Our next model is logistic, with objective function:

$$\begin{aligned} f(\mathbf{w}) &= g(\mathrm{diag}(\mathrm{sign}(\mathbf{y}))D\mathbf{w}) , \\ g(\mathbf{z}) &= \sum_i \ln\left(1 + e^{-z_i}\right) . \end{aligned} \tag{4}$$

The third model is a support vector machine, with objective function:

$$\begin{aligned} f(\mathbf{w}) &= \frac{1}{2}\left(\mathbf{w}^2 - w_0^2\right) + h(\mathrm{diag}(\mathrm{sign}(\mathbf{y}))D\mathbf{w}) , \\ h(\mathbf{z}) &= \sum_i \max\{1 - z_i, 0\} , \end{aligned} \tag{5}$$

where w_0 is the weight corresponding to the offset.

In addition to the choice of model, our Minimax method contains two parameters: N - the number of stones in the last model, and M - the gap between models. N is determined by the desired runtime, and should be chosen as small as possible. For our implementation we chose $N = 78$, 10 stones away from a full board. This choice allowed us to generate and score thousands of training examples with only a few minutes of runtime. Choice of M is more dependent on the game - if the importance of features changes quickly as a function of time, M should be chosen small, otherwise a larger M is preferred. However, because our method uses Minimax with a search depth of M in order to score training examples, there is a practical limit to the size of M.

Once a set of models is learned, a game of Reversi can be played in the standard way - by using Minimax with

some fixed search depth and scoring the leaf nodes by using the models. Because we learn models in discrete intervals, some decision must be made as to how to score intermediate boards. We consider rounding up to the nearest model, as well as linearly interpolating between the two nearest models.

Given these various choices of models and parameters, we select the best by competing the Reversi AIs against each other. In order to generate different games, we make random moves from the starting configuration until we have 16 stones, then allow the AIs to take over. To ensure fairness, the board is then replayed with the colors swapped. Because some starting positions may be very unequal, we expect the win/loss ratio to be artificially pulled towards $\frac{1}{2}$. With this setup, the average stone differential may be a truer judge of quality.

VI. RESULTS

A. Training method

We begin with a comparison of the 3 different model types - linear, logistic, and SVM - discussed in the previous section. 10000 boards with 78 stones are randomly generated, then converted into feature matrix D, and Minimax score vector y. A set of weights is learned for each model - for the linear model, Eq (3) can be minimized exactly, for the logistic model we use batch gradient descent, and for the SVM we apply an ADMM approach [14] to minimize the non-differentiable objective function in Eq (5). Because the number of training samples was far greater than the number of features, we also use the training data to compare the resultant weight vectors. These are compared using two measures: the standard deviation of the predicted score from the actual score:

$$\sigma = \sqrt{\frac{1}{N}\left(D\mathbf{w} - \mathbf{y}\right)^2}\,, \tag{6}$$

and the classification success rate:

$$p = \frac{1}{N}\sum_i \frac{1}{2}\left(\mathrm{sign}(y_i(D\mathbf{w})_i) + 1\right)\,. \tag{7}$$

Because the logistic and SVM models are classifiers, their weight vectors must be rescaled in order to make σ a meaningful comparison. We choose the scaling which minimizes σ.

The results for the 78 stone models are shown in Table I. We find that even with our limited feature set, the models do a fairly good job. When asked to predict whether a board is a winning or losing one, the models get it right more than 80% of the time. For a score which can vary anywhere in the range $[-88, 88]$, a standard deviation of 13 is not bad, as we can see in Fig 3. Although reassuring, these 78 stone results show little variability and do not help in discriminating between the three different methods. In order to do that, we must train the full set of models and evaluate the quality of the Reversi gameplay.

	Linear	Logistic	SVM
σ	13.430	13.470	13.473
p	82.86%	83.04%	83.03%

TABLE I: Comparison of 3 models using 78 stone training data

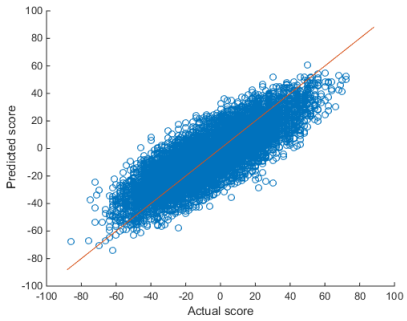

FIG. 3: SVM model predicted score vs actual score.

Three full sets of models were constructed using our Minimax training method with $N = 78$ and $M = 4$. Each model was trained using a random set of 2000 boards which were scored using the results of the previous model. In this way, we learned a board evaluator for $78, 74, 70, \ldots, 6$ stone boards. Given the small spacing, we decided to evaluate intermediate stone boards by rounding up to the nearest model. The linear, logistic, and SVM models were then played against each other many times to determine the best one. We report both the win ratio, calculated as:

$$\mathrm{WR} = \frac{\mathrm{Wins} + \mathrm{Draws}/2}{\mathrm{Games}}\,, \tag{8}$$

and the average stone differential (SD) between the two opponents.

The results can be found in Table II. For each pairing, we played out 4000 16-stone boards twice, swapping colors the second time. Both sides used the same search depth (ply) which we varied from 1 to 5. If this is a valid test of model quality, we expect it to be largely independent of the search depth, and we find this to be the case. Comparing our three different models, we find that SVM outperforms the other two in both win ratio and stone differential. Based on these results we use the SVM classifier for all subsequent models discussed in this paper.

Logistic vs Linear		
Ply	WR	SD
1	.526	2.8
2	.522	1.5
3	.523	1.3
4	.525	1.9
5	.523	1.2

SVM vs Linear		
Ply	WR	SD
1	.534	3.6
2	.544	3.4
3	.545	3.2
4	.567	5.5
5	.562	4.7

SVM vs Logistic		
Ply	WR	SD
1	.507	0.7
2	.534	3.1
3	.529	2.2
4	.552	4.1
5	.536	2.4

TABLE II: Win ratio (WR) and average stone differential (SD) for Reversi matches played with a (Ply) lookahead, using board evaluators trained using linear, logistic, and SVM classifiers. The results are reported for the first player (left of the vs). The error estimates for the WR and SD are .006 and .4 respectively.

B. Score propagation

As discussed earlier, in the limit of perfect models, our Minimax method converges to the true Minimax value for any board. For more realistic models, though, model imperfections will lead to errors in board evaluations. How do these errors affect the training of subsequent models? We can explore this question experimentally near the endgame. To do this, we first train an 83 stone model. Two separate 78 stone models are then trained from the same set of boards, one scored with depth-5 Minimax and the 83 stone model, and the other scored using depth-10 Minimax and the actual endgame score. The first model attempts to learn Minimax values through an intermediary model, the second represents the best possible performance which could have been achieved. In Table III we compare model predictions to the actual Minimax values for both 78 stone models as well as for a simlar setup with an intermediate 82 stone model and two 74 stone models. In both cases, the model-trained version falls short of the ideal, but not by much. We find this unsurprising as intermediate model imperfections can be treated as a noise term in the board scoring. Because our training data includes roughly 100 boards per feature, we expect most of the noise to be averaged out. For the late game models, this does appear to be the case, giving us confidence that endgame scoring information can be usefully propagated back to earlier boards.

78 stone models		
	Model-trained	Ideal
σ	14.08	13.62
p	82.11%	82.27%

76 stone models		
	Model-trained	Ideal
σ	14.70	14.46
p	81.59%	82.76%

TABLE III: Late game model comparison, one trained using an intermediate model, the other trained with the exact Minimax values.

C. Model selection

An important parameter in our Minimax method is M, the span between subsequent models. Small M allows for a better temporal resolution, at the cost of an increased number of models (presumably increasing the noise in earlier board evaluations). We explore this tradeoff by comparing models with $M = 4\,,\, 6\,,\, 8\,,$ and 10. For $M = 4\,,\, 6$, the Minimax method from Section V is used exactly as described. For larger M, however, the runtime becomes impractical - the $M = 4$ models were trained in minutes, the $M = 6$ models took several hours. For larger M, we instead used an iterative, interpolating approach. The N stone model is trained as before, using Minimax and the endgame scores. An $N - M$ stone model is then initialized using the N stone weights. A set of $N - M$ stone boards are generated and scored using a depth-$M/2$ Minimax search with terminal boards scored using the average weight vector of the N and $N - M$ stone models (initially the same). A new $N - M$ stone model is learned from this data, and the whole process is repeated several times until the $N - M$ weights converge.

Before comparing the models against each other, we must first decide how to score boards which fall between two models. In Table IV we examine two methods: using the larger stone model, and linearly interpolating between the smaller and larger models. As before, simulations are run using a search depth of $1-5$; the full results are given in the Appendix, we report here the median win ratio and the median stone differential for each matchup.

Players	WR	SD
4(L) vs 4(I)	.480	-1.2
6(L) vs 6(I)	.484	-0.9
8(L) vs 8(I)	.472	-1.8
10(L) vs 10(I)	.469	-1.9

TABLE IV: Comparison of 2 different methods for scoring intermediate boards (L - use larger stone model, I - interpolate between the two). Results are reported for the (L) player.

In all cases, interpolation gave a better result, even for models spaced only 4 stones apart. Therefore we use interpolation for all subsequent results discussed in this paper.

Finally, we examine the effect of M by competing the different M models against one another. The full $1 - 5$ ply results are given in the Appendix, the median values are shown in Table V. For our 88 stone board, it appears that larger M gives better results.

D. Minimax vs game data

Finally, we compare our method against the most common method of training a board evaluator - game data. Since none exists for the 88 stone board, we switch to the

Win ratio				
M	4	6	8	10
4	-	.514	.473	.478
6	.486	-	.472	.467
8	.527	.528	-	.488
10	.522	.533	.512	-

Stone differential				
M	4	6	8	10
4	-	1.5	-1.2	-2.3
6	-1.5	-	-2.4	-3.3
8	1.2	2.4	-	-1.7
10	2.3	3.3	1.7	-

TABLE V: Round robin results. Scores are reported for the player in the first column.

standard 8×8, 64 stone board. Based off our previous 88 stone results, both model sets are trained using an SVM classifier, and intermediate boards are scored using interpolation. Both model sets use the same features, but use different training data. The first model set is trained using our Minimax method with $N = 54$, $M = 4$, giving rise to 54, 50, 46...6 stone models. The second model set is trained using the WTHOR database - a collection of top-level Reversi tournament matches [15]. For each of the 54, 50, 46...6 stone boards we identify 2000 examples, scoring each with the stone differential at the end of the match, and a corresponding model is trained. Because the 88 stone results showed a preference for larger M, we duplicated this experiment for a set of models spaced 10 stones apart. The Minimax and game data models were then played against each other; the results are shown in Table VI.

Players	WR	SD
4(M) vs 4(G)	.574	5.4
10(M) vs 10(G)	.575	5.8

TABLE VI: Comparison of 2 different methods for model training (M - Minimax, G - game data). Results are reported for the (M) player.

In addition to the head-to-head matches, we tested our two programs against several other AIs. Because of the relatively simple model used, we expect our AIs to play at a strong, but not unbeatable level, and therefore two opponents, *Springfrog* [16], and *Webversi* [17] were chosen from among online, human-playable Reversi AIs. The last opponent, *WZebra* [18], is a world-class program with features similar to *Logistello*. For the first three matches, we used the $M = 4$ models with a search depth of 8. Because *WZebra* was set to only use a 4-ply lookahead, for a direct comparison we played a final match, restricting our search depth to 4 as well. The scores of these matches are given in Table VII.

	Springfrog	Webversi	WZebra	WZebra
Minimax	57-6	52-10	37-27	20-44
Game data	48-16	31-33	29-35	15-49

TABLE VII: Match scores between our $M = 4$ Minimax/game data trained models and various opponents. Our programs used an 8-ply lookahead for the first 3 matches, and a 4-ply lookahead for the final one.

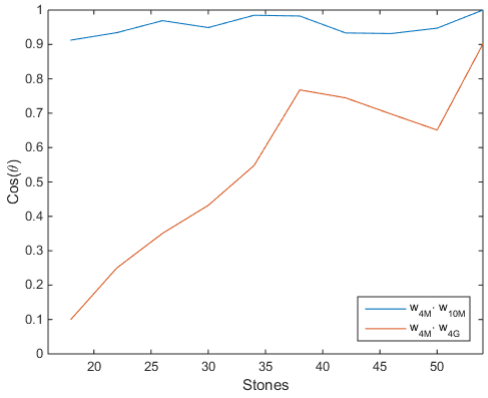

FIG. 4: Similarity comparison between (M) Minimax trained models and (G) game data trained models.

VII. DISCUSSION

We have demonstrated that models trained using our Minimax method can outperform models trained on game data. In a series of Reversi games, our Minimax models consistently outscored their game data trained opponents by a 5 stone margin. This trend held against other opponents as well, with the Minimax trained model achieving better results. As stated in Section IV, we believe this is due to flaws in human gameplay and we suspect that most of our gains come in the middle and early game stages, where the optimal gameplay is unclear. Some evidence for this is found in Fig 4, where we plot the dot product of normalized weight vectors as a function of the number of stones. For the 2 Minimax models, one with a spacing of 4, the other with a spacing of 10, we see that despite being trained on different boards and with different spacing, the two model sets appear to be learning very similar weight vectors throughout the game. The game data model, on the other hand, appears to find a similar set of weights near the end of the game, but this similarity diminishes in the early stages; by the 16 stone model, the two weight vectors are nearly othogonal! Based off of the overall performance, it seems likely that our Minimax method does a better job of learning the early game models.

One of the main advantages to our Minimax method is its independence from expert knowledge. All board evaluations are drawn (indirectly) from endgame scoring, rather than relying on potentially imperfect information. Not only does this appear to be more accurate, it also allows for the quick generation of robust computer players for uncommon game variants (such as our 88 stone board) as well as completely new games. However, expert knowledge is still useful for the generation of quality models and features, a foundation upon which

our method relies. A poor set of features may affect the Minimax trained models more significantly than game-trained models, as board evaluation mistakes can be compounded. Conversely, though, any model improvements should lead to a larger overall improvement for the Minimax trained player as opposed to the game data trained player.

Although the results reported here validate our Minimax method, we believe there is still room for improvement. One such improvement would be in the generation of training boards. Currently these are created by making random moves from the initial starting position. We believe a better method would be to use a previously learned set of models to play from a random starting configuration. This way the training boards will be drawn from a probability distribution similar to one the AI might face in a competition, and the models will be more accurate for this subset of boards. For Reversi, an example of how this might improve things would be the feature corresponding to the corner square. This is a powerful square, as it can't be flipped, and can be used to make other stones stable as well. In a set of boards generated from random moves, however, possession of the corner will not seem as significant, as its advantage will often be squandered through senseless, random moves. A simple model will not be able to differentiate between a corner square possession used well and a corner square possession used poorly, and therefore the overall importance of the corner will be supressed. A set of boards played by a trained Reversi AI, on the other hand, will generally use the corners well, making this feature more significant. We have elected not to make this improvement for runtime reasons. Many training examples are required for our method, using a Reversi AI with a non-trivial search depth to generate these would require several days of computation. It is doable, however, and it would be interesting to see how much of an improvement this would deliver.

VIII. CONCLUSION

We have introduced a new method for training board evaluation models. Rather than game data or self-play, we use endgame scoring and the Minimax algorithm along with a series of models which propagate information about the endgame backwards to earlier game stages. It is a simple approach with only one tunable parameter: M, the separation between adjacent models. When applied to our relatively simple Reversi model, we found that the best results were given by using an SVM classifier, as well as evaluating intermediate board positions using linear interpolation. Large M seemed to be preferred in this case; presumably any feature variation on smaller timescales was less important than the gains produced by learning fewer models. Finally we trained both Minimax and game data based models on the standard 8×8 Reversi board and played the resulting AIs against each other. Although game data training is the standard approach, our Minimax model outperformed the game data model by a substantial 5 stone margin, demonstrating the potential of this new approach.

APPENDIX A: REVERSI GAME RESULTS

Reversi game results were generated by playing 4000 randomly generated 16 stone boards. Players switch colors and replay boards for a total of 8000 games. Scores are reported for the first player (left of the vs). Standard error estimation using the variance suggest the WR measurements reported here are accurate to within .006 and SD within 0.4.

4(L) vs 4(I)		
Ply	WR	SD
1	.487	-1.2
2	.485	-0.9
3	.479	-1.4
4	.480	-1.2
5	.479	-1.0

6(L) vs 6(I)		
Ply	WR	SD
1	.473	-2.0
2	.474	-1.9
3	.484	-0.9
4	.490	-0.4
5	.498	0.9

8(L) vs 8(I)		
Ply	WR	SD
1	.466	-2.5
2	.463	-2.8
3	.472	-1.8
4	.481	-1.0
5	.486	-0.1

10(L) vs 10(I)		
Ply	WR	SD
1	.463	-2.8
2	.462	-2.8
3	.469	-1.9
4	.479	-0.7
5	.498	0.9

TABLE VIII: Interpolation results

4 vs 6		
Ply	WR	SD
1	.514	1.5
2	.498	-0.2
3	.500	0.1
4	.522	2.3
5	.522	2.0

4 vs 8		
Ply	WR	SD
1	.470	-2.0
2	.469	-2.0
3	.491	0.3
4	.473	-1.2
5	.483	-0.9

4 vs 10		
Ply	WR	SD
1	.486	-1.3
2	.457	-4.2
3	.473	-2.4
4	.478	-2.3
5	.481	-1.6

6 vs 8		
Ply	WR	SD
1	.479	-2.4
2	.472	-1.9
3	.506	0.5
4	.461	-3.2
5	.460	-3.0

6 vs 10		
Ply	WR	SD
1	.472	-3.2
2	.466	-3.3
3	.487	-1.6
4	.450	-5.1
5	.467	-3.3

8 vs 10		
Ply	WR	SD
1	.498	-0.6
2	.483	-1.9
3	.483	-1.9
4	.488	-1.7
5	.493	-1.2

TABLE IX: 88 stone round robin results

4(M) vs 4(G)		
Ply	WR	SD
1	.574	5.4
2	.537	3.2
3	.570	5.7
4	.580	5.7
5	.592	6.8

10(M) vs 10(G)		
Ply	WR	SD
1	.562	4.7
2	.563	4.5
3	.575	5.8
4	.590	6.1
5	.612	7.6

TABLE X: 64 stone results

[1] D. E. Knuth and R. W. Moore, Artificial Intelligence **6**, 293 (1975).
[2] P. Rosenbloom, Artificial Intelligence **19**, 279 (1982).
[3] K.-F. Lee and S. Mahajan (1986).
[4] K.-F. Lee and S. Mahajan, Artificial Intelligence **36**, 1 (1988).
[5] M. Buro, in *19th Annual Conference Gesellschaft für Klassifikation eV* (Citeseer, 1995), pp. 1–3.
[6] M. Buro, in *ICCA Journal* (Citeseer, 1995).
[7] M. Buro, Journal of Artificial Intelligence Research pp. 373–382 (1995).
[8] G. Tesauro, Communications of the ACM **38**, 58 (1995).
[9] I. Ghory, Department of Computer Science, University of Bristol, Tech. Rep (2004).
[10] A. Leouski, *Learning of position evaluation in the game of othello* (Department of Computer Science, 1995).
[11] J.-F. Isabelle, Ph.D. thesis (1995).
[12] T. Yoshioka and S. Ishii, IEICE TRANSACTIONS on Information and Systems **82**, 1618 (1999).
[13] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming* (Athena Scientific, 1996), 1st ed., ISBN 1886529108.
[14] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, Found. Trends Mach. Learn. **3**, 1 (2011), ISSN 1935-8237, URL `http://dx.doi.org/10.1561/2200000016`.
[15] F. F. of Othello, *Wthor database* (2015), online; accessed 19-May-2015, URL `http://www.ffothello.org/informatique/la-base-wthor`.
[16] Springfrog.com, *Springfrog* (2015), online; accessed 26-May-2015, URL `http://www.springfrog.com/games/reversi`.
[17] F. LaRosa, *Webversi* (2015), online; accessed 26-May-2015, URL `http://webversi.com`.
[18] G. Andersson, *Wzebra* (2015), online; accessed 26-May-2015, URL `http://radagast.se/othello`.