Learning Othello Board Evaluation with Artificial Neural Network

Théo Verhelst

January 9, 2018

Abstract

Othello (also named Reversi) is a board game that gained interest in research for the simplicity of its rules and its interesting branching factor. Numerous heuristics for board evaluation have been developed over the past decades, but there is a growing trend towards the use of artificial neural networks (ANN) for this purpose. We implemented minimax search algorithm with alpha-beta pruning and simple heuristic in order to allow a human user to play Othello against a competitive artificial player. We then used a multilayer neural network to learn a board evaluation function, by using temporal difference learning (TDL) and self-play. The ANN is used as the heuristic function of minimax search with a ply depth of 1. Our ANN outperforms minimax search at ply depth of 6 with carefully handcrafted heuristic.

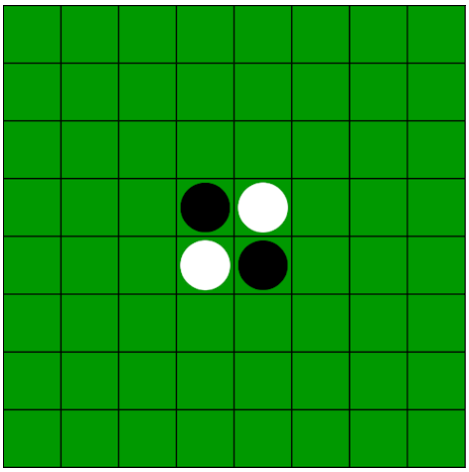

Figure 1: *Initial board of the Othello game. Credits:* `http://mnemstudio.org/game-reversi-intro.htm`

1 Introduction

Our main focus was to study the use of artificial neural network (ANN) as heuristic evaluation for minimax search. We chose to center the search on the game Othello, since it has been well studied in the field of AI and machine learning. Moreover, it has an interesting branching factor (around 7 on average [1]), which is not as big as in chess, but still big enough to prevent us from doing exhaustive tree search. We came across the article [2], which is a great resource about state-of-the-art Othello board evaluation with ANN. In this article, Binkly et al. claim that a 1-ply search is sufficient to compete against strong Othello players, if an efficient neural network architecture is used. We decided to implement a basic architecture, which, hopefully, will result in a decent Othello player.

An important part of the learning process in this kind of experiment is the way data is extracted from game history, in order to be fed into the ANN. For this purpose, Sutton et al. developed the Temporal Difference Learning (TDL) [3]. TDL is used when predictions must be done in sequence, and when the target value is available at the end of the sequence of predictions. We use the current prediction to correct past ones, by supposing that more recent prediction are more accurate, since we are closer in time to the outcome.

The rest of this document is outlined as follow. Section 2 briefly explains the rules of the Othello game. The TDL procedure is exposed more precisely in section 3. In section 4, the details of how ANNs are used for the purpose of board evaluation are investigated. The setup of the experiment is given in section 5, and its results are given in section 6.

2 Othello game

Othello is a two-players board game, played on a grid of 8 by 8 cells. The player who plays first has black pieces, and the second player has white pieces. The initial board is shown in figure 1. Each turn, a player puts one of its piece on an empty cell, in such a way as to capture at least one of the opponent's pieces. A line of pieces is captured if it is bounded by opponent's pieces at both ends. If no move leads to a capture, the game is over and the player with the most pieces on the board wins.

3 Temporal Difference Learning

A major problem arises when one wants to use machine learning to play complex board games such as Othello. A game is composed from a sequence of states, and the role of the evaluation function is to learn whether a given game state is likely to lead to a win. One could therefore train the model to predict a win for all boards encountered in a game ending by a win, and similarly to predict a loss for all boards encountered in a game ending by a loss. The main problem with this approach is that there is a lot of uncertainty in games such as Othello, and none of the players are perfect. Therefore, the correlation between a state in the beginning of a game and the outcome of this game may be low [2].

Temporal difference learning (TDL) [3] is a solution to this issue. According to [2, 1], the method TD(0), which is a special case of the TD(λ) class of temporal difference methods, is the most appropriate to learn from Othello games. Let S be the set of all possible board states, $(s_1, s_2, \ldots, s_n) \in S^n$ the sequence of states of a given game, and z the outcome of the game. Consider the sequence of states evaluated by the first player. Since we are using 1-ply search, this sequence is $(s_2, s_4, \ldots)$. Let us rename this sequence $(v_1, v_2, \ldots, v_m)$. The evaluation of a state v_t by the first player is denoted $f(w, v_t)$, where w is the weight vector of its evaluation function. The classical approach to TD(0) is to compute a weight increment at each turn using the formula given in [3]:

$$\Delta w_t = \alpha(f(w, v_{t+1}) - f(w, v_t))\nabla_w f(w, v_t) \quad \forall t \in \{1, \ldots, m\} \tag{1}$$

where $f(w, v_{m+1}) := z$, and α is the learning rate. We have then to update w at the end of the game, when the outcome z is known

$$w \leftarrow w + \sum_{t=1}^{m} \Delta w_t \tag{2}$$

However, this approach was difficult to apply to our study. Indeed, we used a neural network library for Java called Neuroph, and it does not include temporal difference learning out of the box. A lot more effort would therefore have been needed to implement this learning rule. Instead, we used the approach explained in [4], which consist of learning to predict the evaluation of current board $f(w, v_t)$ when the previous board state v_{t-1} is given as input. The pairs $(v_{t-1}, f(w, v_t))$ are stored while the game is running, and the model learns from them once the game is finished. This approach and the one given in [3] are equivalent, and the difference is in fact an implementation detail.

4 Neural Network for Board Evaluation

Numerous ANN architectures have been studied for the purpose of learning Othello board evaluation [2, 1]. The one we chose is the first one described in [1]. We selected this architecture for its simplicity. It consists solely of 64 inputs units, one for each of the board cells, one hidden layer of 50 units, and one output unit. The board is encoded by a number for each board cell, where 0 accounts for an empty cell, 1 for a cell with the player's piece, and -1 for a cell with the opponent's piece. The game outcome z is encoded similarly: 1 for a win, -1 for a loss and 0 for a draw.

According to the results of [2], a network input consisting of 2 or 3 input units per board cell outperforms the approach described above, given equivalent ANN architectures. Therefore, we used the following board encoding described in [2]: $(1, -1)$ for the player's pieces, $(-1, 1)$ for the opponent's pieces and $(-1, -1)$ for an empty cell.

There is some other features that we could have used in order to improve our ANN, most of them are described in [2]: weight sharing to account for board symmetry, symmetry removal, spatial processing layer, multiple hidden layers, exhaustive minimax search at the end of the game. However, we chose not to implement any of them for time constraint reasons.

We used a naive solution to account for board symmetry, which has not been proposed in any of the referenced articles. When a board is proposed to the ANN for learning, all of the 8 symmetries of this board [2] are generated, and they are also proposed for learning to the ANN with the same target value. This method resulted in a slight performance increase.

5 Experimental Setup

We followed the training methodology given in [2], which consists of making two ANNs playing against each other, learning from the games according to the TD(0) method, and frequently evaluating their performance against a reference player. The reference player we used is described in [2] and [5]. [1]. It uses minimax search at ply depth of 3 with a handcrafted heuristic function, which is defined for some board

[1] Blinkly et al. claim in [2] that this reference player originated from [5]. However, the table in figure 2 is not to be found in [5], while the equation 3 is indeed present. Therefore, we give the credit of this reference player to both articles.

1	100	-25	10	5	5	10	-25	100
	-25	-25	2	2	2	2	-25	-25
	10	2	5	1	1	5	2	10
	5	2	1	2	2	1	2	5
	5	2	1	2	2	1	2	5
	10	2	5	1	1	5	2	10
	-25	-25	2	2	2	2	-25	-25
57	100	-25	10	5	5	10	-25	100

8

64

Figure 2: *Weights of the heuristic function used by the reference player. Credits: [2].*

state x by

$$f(x) = \sum_{i=1}^{64} w_i x_i + N(0, \sigma_{noise}) \tag{3}$$

where $N(0, \sigma_{noise})$ is Gaussian noise with mean 0 and standard deviation σ_{noise}, and w_i is defined according to the table in figure 2.

We trained two ANNs against each other in 45,000 successive games, and one of them played 50 games against the reference player every 150 training games, in order to evaluate the evolution of the ANNs performance.

In the learning process, ϵ-decreasing move choice policy [6] has been used. In the first game, the ANNs choose the estimated optimal move with probability $1 - \epsilon = 0.9$, and selects a random one with probability $\epsilon = 0.1$. As the learning process runs, the value of ϵ decreases linearly, and finally reaches 0 in the last game of the training process.

6 Results and Discussion

Results of the experiment are shown in figure 3 and 4. The winning rate evolution in figure 3 is very noisy, we thus filtered the sequence with a low-pass filter in figure 4, in order to see more clearly the progression. We first ran the whole learning process with a learning rate $\alpha = 0.1$. We noticed that the the performance was not stable and did not evolve noticeably. Therefore, we found two other promising values by grid search, $\alpha = 0.02$ and $\alpha = 0.001$. While $\alpha = 0.02$ leads to quicker learning than $\alpha = 0.001$, its also results in less stable performance. When comparing the three curves, it is clear that $\alpha = 0.1$ results in an overshooting of the optimal weights, and therefore the convergence was not attained.

Ply depth	ANN 1	ANN 2
1	0.65	0.53
2	0.58	0.65
3	0.65	0.7
4	0.69	0.56
5	0.81	0.71
6	0.76	0.7

Table 1: *Winning rate of both trained ANN with $\alpha = 0.02$ against the reference player with various ply depths.*

It seems that our ANN player finally outperformed the minimax player by reaching a winning rate greater than 0.5. That indicates that the neural network learned from the raw board input features at least as efficient as a carefully handcrafted heuristic developed by experts.

Another important aspect of these results is that we were not able to formally assess the influence of various features we implemented, such as the number of input per board cell, the number of hidden neurons, the training procedure, and so forth. We relied on the results of previous studies when it came to decide whether or not to use them. Ideally, we would had run the same training process with and without them, in order to make sure that they are relevant to our problem.

Once the ANNs has been trained, we ran test games between both neural network players and the reference player at various ply depth. Each winning rate has been calculated from 100 test games. Surprisingly, the ply depth of the opponent does not seem to influence the winning rate of the ANN player. This may be explained by the fact that we use TDL methods, which may be thought as having an effect approximating a minimax search with infinite ply depth. Indeed, by indirectly back-propagating the final outcome of a game to all boards in the history of the game, the neural network learns which kind of board usually leads to a win, and conversely. This approximation becomes better as the game comes closer to its end.

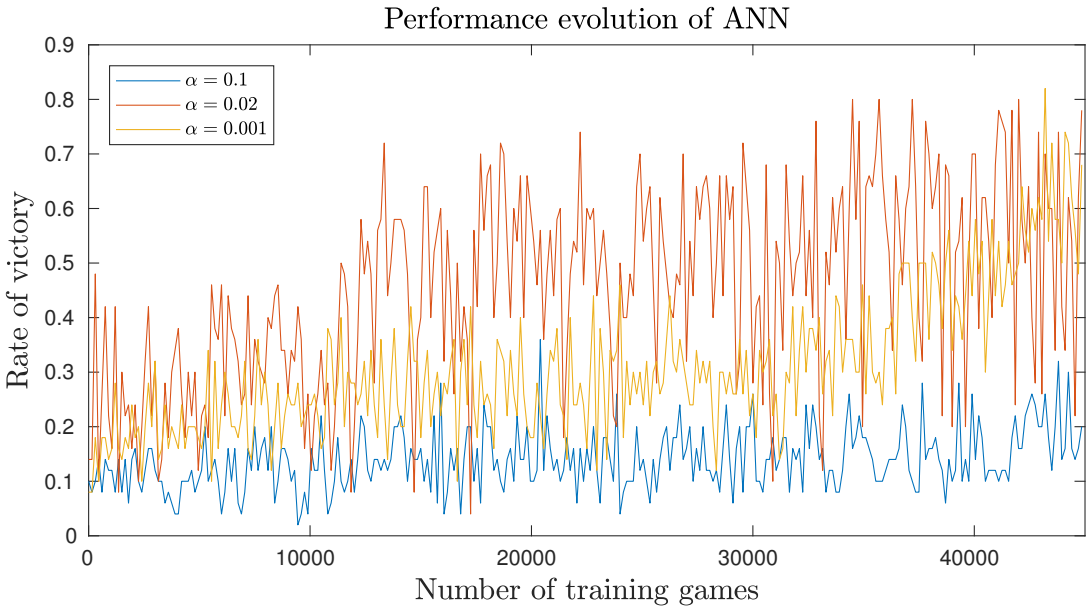

Figure 3: *Evolution of the winning rate against the reference player, for various values of the learning rate.*

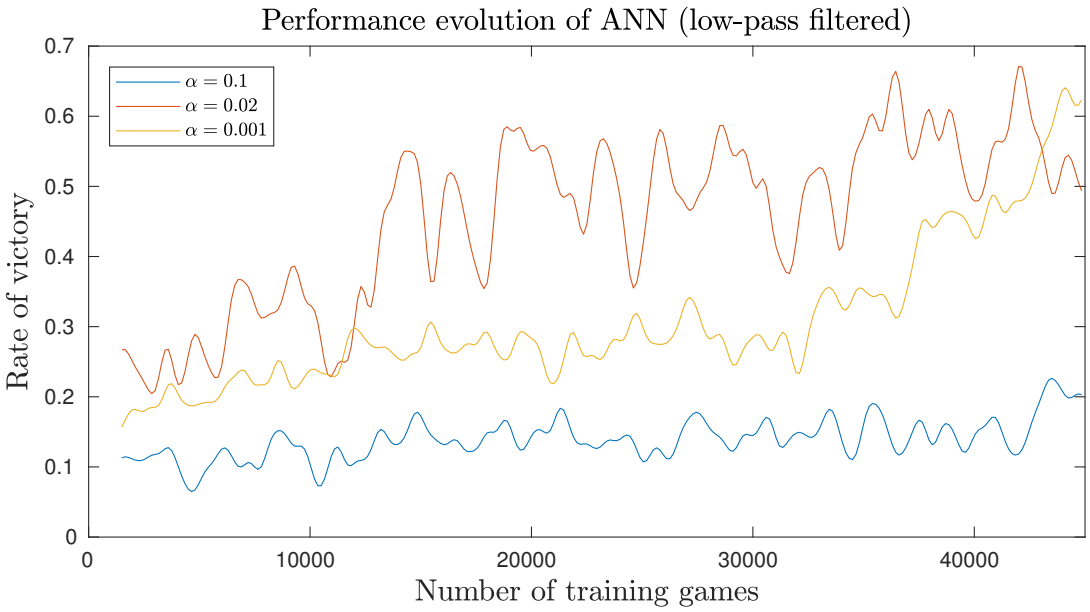

Figure 4: *Evolution of the winning rate against the reference player, for various values of the learning rate. The data has been filtered with a Gaussian filter with window of width 825.*

References

[1] A. V. Leouski and P. E. Utgoff, "What a neural network can learn about othello," 03 1996.

[2] K. J. Binkley, K. Seehart, and M. Hagiwara, "A study of artificial neural network architectures for othello evaluation functions," vol. 22, pp. 461–471, 01 2007.

[3] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, pp. 9–44, Aug 1988.

[4] M. V. D. Steeg, M. M. Drugan, and M. Wiering, "Temporal difference learning for the game tic-tac-toe 3d: Applying structure to neural networks," in *2015 IEEE Symposium Series on Computational Intelligence*, pp. 564–570, Dec 2015.

[5] T. Yoshioka, S. Ishii, and M. Ito, "Strategy acquisition for the game "othello" based on reinforcement learning," vol. E82-D, 05 1999.

[6] S. L. Scott, "A modern bayesian look at the multi-armed bandit," *Applied Stochastic Models in Business and Industry*, vol. 26, no. 6, pp. 639–658, 2010.