Learning of Position Evaluation in the Game of Othello

Anton Leouski

Master's Project: CMPSCI 701
Department of Computer Science
University of Massachusetts
Amherst, Massachusetts 01003

leouski@cs.umass.edu

January 20, 1995

Abstract

Conventional Othello programs are based on a thorough analysis of the game, crafting sophisticated evaluation functions and supervised learning techniques with use of large expert-labeled game databases. This paper presents an alternative by training a neural network to evaluate Othello positions via temporal difference (TD) learning. The approach is based on network architecture that reflects the spatial and temporal organization of the domain. The network begins as a random network and by simply playing against itself achieves an intermediate level of play.

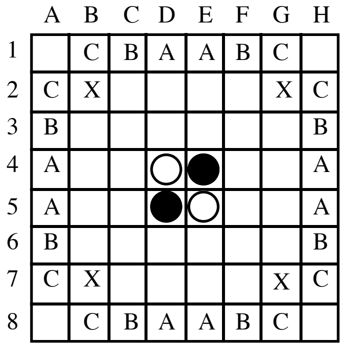
1. Introduction

Othello is the third incarnation of the old Japanese board game, developed in its current form in 1974 [16]. It is mostly attractive because of the simplicity of its rules. One may start playing after a minute of introduction, and still the game has enough complexity to leave a dedicated person with years to master. The difficulty to envision dramatic changes of the disk patterns on the board makes Othello quite a challenge for human players. Computers seem to have an equivocal advantage in this domain. The best computer programs are based on a thorough analysis of the game, crafting a set of high-level concepts, implementing very sophisticated searching techniques. These programs usually incorporate learning methods with use of huge expert-labeled game databases.

This paper describes a viable alternative to the conventional methods by training a neural network to evaluate Othello positions via temporal difference (TD) learning. The network architecture incorporates some of the spatial and temporal properties of the domain. Starting from scratch, after a few thousand training games the network achieves a rather decent level of play.

The paper outlines description of the game of Othello and its rules (Section 2); reviews some of the major research done in this domain (Section 3); presents the architecture of the network used in the experiments (Section 4). Section 5 describes how the network was trained; Section 6 summarizes the major results; Section 7 projects some ideas for the future, and Section 8 contains some final remarks.

2. The Game of Othello



	A	B	C	D	E	F	G	H
1		C	B	A	A	B	C	
2	C	X					X	C
3	B							B
4	A			○	●			A
5	A			●	○			A
6	B							B
7	C	X					X	C
8		C	B	A	A	B	C	

Fig. 1. Shows the initial Othello board setup and the standard names of the squares.

Othello is played on an 8-by-8 grid, using dual-colored disks. Every disk has one white and one black side. Like chess, it is a deterministic, perfect information, zero-sum game of strategy between two players, black and white. Black opens the game from the initial board configuration shown in Fig. 1 [9]. A *legal move* for a player is a placement of a piece on the board resulting in the capture of one or more opponent's pieces. A *capture* occurs when a player places a piece of his color in a blank (empty) square adjacent to a line of one or more pieces of the opposing color followed by a piece of his color. Captured disks are flipped to the captor's color. Fig. 2(a) contains a board with legal moves for white to e2, d3, c4, and e6. Fig. 2(b) shows the board after white moves to c4.

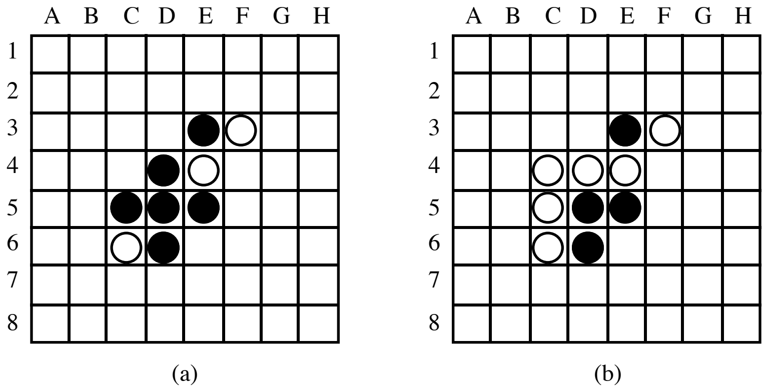
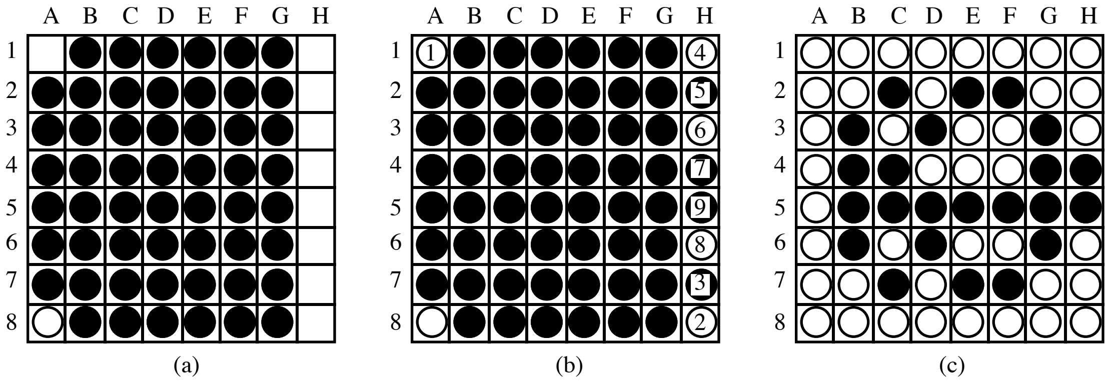


Fig. 2. (a) shows a sample board with legal moves (for white) to e2, d3, c4, and e6; (b) shows the board after white plays to c4.

Play continues until neither player has a legal move, which usually happens when the board is completely filled. At the end of the game, the pieces of each color are counted, and the player with the most pieces is declared the winner.

The beauty of Othello is that just one move on the board may change the game situation very dramatically. From the Fig. 3(a) one may assume that white is completely lost, it has only one piece left, but simple analysis in Fig. 3(b) shows that white actually wins 23x41.



(a) (b) (c)

Fig. 3. (a) white seems very close to loosing the game; (b) shows the record of the next 9 moves; (c) shows the final position.

Standard Othello notation is different from the one used in chess that numbers (1-8) indicating rows go from top to bottom. Some of the more important squares also have names. For example, the square diagonally next to a corner is called X-square. Fig. 1 shows the standard names of the Othello squares.

3. Previous Research

The game of Othello has received great attention within Computer Science for more than ten years. The following is the far from complete outline of several pieces of research work in this domain.

IAGO

It was Paul Rosenbloom [9] who pointed out that although the game of Othello has an average branching factor 5 and limited length (less than 64 moves) it still cannot be solved exactly and has a great deal of complexity to be a subject for scientific analysis. Rosenbloom analyzed the game of Othello into a pair of major strategic concepts (stable territory and mobility), each decomposable into sub-concepts. Quantitative representations of these concepts were combined in a single evaluation function. Together with the α-β search algorithm, interative deepening, and move ordering, this function formed the basis of Rosenbloom's program IAGO.

Although IAGO displayed a World Class performance it did have several drawbacks. First, the concepts set used by the program is rather limited [15]. Second, the concepts, also called *features*, were assumed independent and therefore combined into a linear evaluation function. Third, the application coefficients in the evaluation function were selected by hand, which left a significant margin for error. Fourth, IAGO used a single evaluation function for the whole game, though it is now well known that different strategies are needed for different stages of the game.

BILL

K.-F. Lee and S. Mahajan [6, 7] addressed these drawbacks by creating a program named BILL, which used a slightly extended set of features. These feature representations were significantly improved through use of pre-computed tables that allowed BILL to recognize hundreds of thousands of patterns in constant time. The authors applied Bayesian learning to combine concepts in BILL's evaluation function, which directly estimates the probability of winning. BILL learned several evaluation functions, one for every move between the 25th and 48th, inclusive. These evaluation functions were trained using a large database of 3000 games created by an earlier version of the program.

These properties and improved search techniques and timing algorithms allowed BILL to surpass completely IAGO, but as I mentioned, BILL requires a big game database for training, and the quality of the evaluation function completely depends on the quality of this database. BILL's evaluation function is a quadratic polynomial, which takes into account linear inter-relationships between features; the question of existence of more complex interactions among these concepts remains open.

Genetic algorithms

A different approach was elaborated by D. Moriarty and R. Miikkulainen [8]. They evolved a population of neural networks using a genetic algorithm to evaluate possible moves. Every network sees the current board configuration as its input and indicates the goodness of each possible move as the output. In other words instead of searching through the possible game scenarios for the best move, the neural networks rely on pattern recognition in deciding which move appears the most promising.

The interesting point is that such evolving neural networks or creatures were required to differentiate among all possible moves, both legal and illegal ones, then the best legal move was chosen to continue the game. The results showed that fixed architecture networks were unable to learn any but trivial strategy, when the creatures with a mutating architecture managed to elaborate the concept of mobility.

The weakness of this approach is that the creatures need to be evolved against another Othello player and though the creatures finally outperformed their opponent, the quality of the final networks is proportional to the quality of their opponent.

Neural Networks and Temporal Difference

We have seen that the conventional approach to this kind of problem is to select an evaluation function that will map a particular set of features to a numeric value, then use this function together with a standard search technique to select consecutive moves.

The pattern recognition component inherent in Othello is amenable to connectionist methods. Supervised backpropagation networks might have been applied to the game but would have faced a bottleneck in the training data. Someone would need to provide a significant-sized collection of labeled game records. An alternative approach based on the TD(λ) predictive algorithm [11] has been proposed. This technique was successfully applied to the game of backgammon by G. Tesauro [12]. The advantage of this method is that a neural network can be trained while playing only itself and does not require either precompiled training data or a good opponent.

Tesauro's TD-Gammon program uses a backpropagation network to map preselected features of the board position to an output reflecting probability of winning for the player to move. An evaluation function trained by TD(0) together with a full two-ply lookahead to pick the estimated best move has made TD-Gammon competitive with the best human players in the world [13].

A straightforward adaptation of Tesauro's approach to the Othello domain has been investigated [14]. A fully-interconnected network with one hidden layer of 50 units was trained for a period of 30,000 games using raw board positions as input. Although the network quickly learned importance of the corner squares, it had little knowledge of how to protect them. The network had a tendency to take X-squares early in the game, a tactic which is closely associated with losing.

However, it appears that efficiency of learning can be vastly improved through use of an appropriate network structure. I found out that incorporating domain regularities into the network architecture leads to a gain in performance and learning speed. This is the topic of the next sections.

4. Network architecture

Consider a typical connectionist network that is being trained to evaluate game states just by using a raw board representation. The network is required to learn whatever set of features it might need. The complexity of this task may be significantly reduced by exploiting *a priori* properties of the Othello domain. The disc patterns on the Othello board retain their properties under color inversion, board rotation and reflection.

Color inversion means that if someone flips all discs on the board and changes the player whose turn it is to move, it will result in the equivalent position from the other player's prospective. This property is embedded into the network architecture by encoding the input using +1 for black and -1 for white.

The Othello board is invariant with respect to reflection and rotation symmetry of the square. This symmetry was incorporated into the network by appropriate weight sharing and summing of derivatives. Weight sharing was described by D. E. Rumelhart *et al.* [10] and successfully used by Y. Le Cun *et al.* [5].

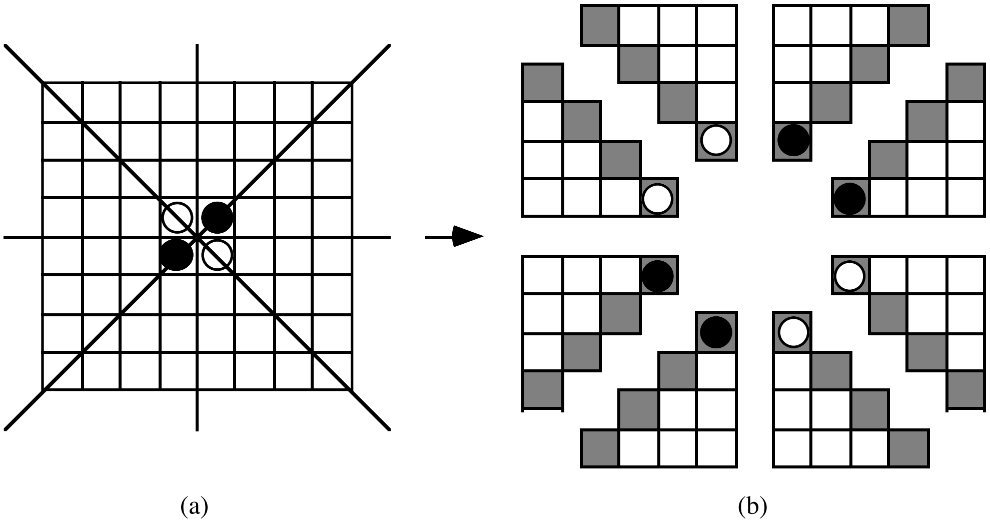


Fig. 4. (a) shows how an Othello board is divided by 8 triangles;
(b) shows the resulting triangles.

The network has 64 input units corresponding to 64 squares on the Othello board. Output value of a unit is 1, -1, or 0, depending on whether the corresponding board square is occupied by black or white disc or is empty. Consider an Othello board (Fig. 4(a)). Let us break the board into 8 pieces by straight lines as shown on Fig. 4(a). You may see the result on Fig. 4(b). The result is eight "triangles"; every one consists of ten board squares.

The diagonal squares are shaded to emphasize the point that these squares are shared between two triangles adjacent to the diagonal.

The input units corresponding to a triangle are connected to one unit from the next layout (Fig. 5(a)), so eight triangles are connected to the eight units from the next layout. Together, eight such units form a plane that Y. Le Cun *et al.* [5] call a *feature map* (Fig. 5(b)).

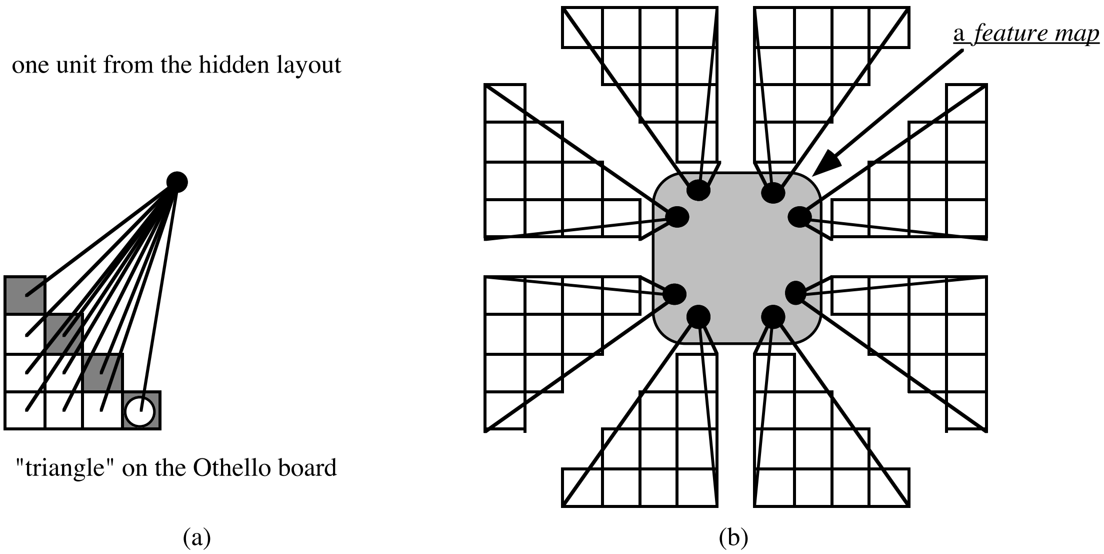

one unit from the hidden layout
"triangle" on the Othello board
(a)
a *feature map*
(b)

Fig. 5. (a) shows connections from 10 input units corresponding to a board "triangle" to one unit from the hidden layout; (b) shows eight "triangles" give input to the eight units in a feature map.

Now, suppose we are training the network to recognize a particular pattern or feature on one such triangle. It is obvious that the kind of feature that is important at one place on the board is likely to be important in other places, at other triangles. Therefore, corresponding connections on each unit in a feature map are constrained to have the same weights. This is achieved by weight sharing, for example, all connections from a given feature map to B-squares have the same single weight value.

Several feature maps create the hidden layout of the network. All units in the hidden layout are connected to the one output unit. Also, every unit from hidden and output layers has a bias unit (with constant output 1) connected to it. Finally, every unit from these two layers has a nonlinear activation function (squashing function, see Fig. 6).

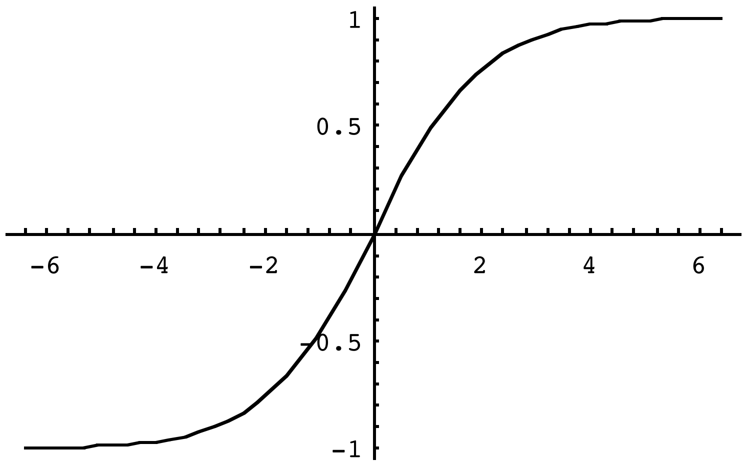


Fig. 6 the nonlinear activation function used in the network

$$Squash(x) = \frac{2}{1+e^{-x}} - 1$$

Therefore, the network used in experiments had one input layer, one hidden layer and one output layer. The input layer had 64 units, the hidden layer was composed from 12 feature maps (96 units), and the output layer had one unit. Note, that the conventional fully-interconnected network would have (64+1)x96 + (96+1)x1 = 6337 different connections and parameters, when the network of the suggested architecture has (10+1)x96 + (96+1)x1 = 1153 connections and (10x12+1x96) + (96+1)x1 = 313 independent parameters. This makes all computations in the network nearly five times faster.

5. Training Process

The network weights were initialized with random numbers uniformly distributed in the interval [-1, +1]. During the training process the program was played against an opponent. The following formula was used to update the network weights:

$$\Delta w_t = \alpha\left(P_{t+1} - P_t\right)\nabla_w P_t$$

Here the α is the learning rate. A value close to 0.1 is generally used, higher learning rates degrade performance, whereas lower rates reduce fluctuations in performance (see [11, 12]). I set α=0.05[1]. The P_t is the current prediction, the output of the network given the current board state. The next prediction, the P_{t+1} is the result of the minimax search

[1] λ was set to 0. So, the TD(0) algorithm was implemented.

performed to the depth of 2. At the end of the game, when there are fewer than 11 empty squares left an exhaustive minimax search is performed and the P_{t+1} is the actual margin divided by 10 and squashed with the same nonlinear function used in the network:

$$P_t = Squash\left(\frac{\#\,ofNetDiscs - \#\,ofOppDiscs}{10}\right)$$

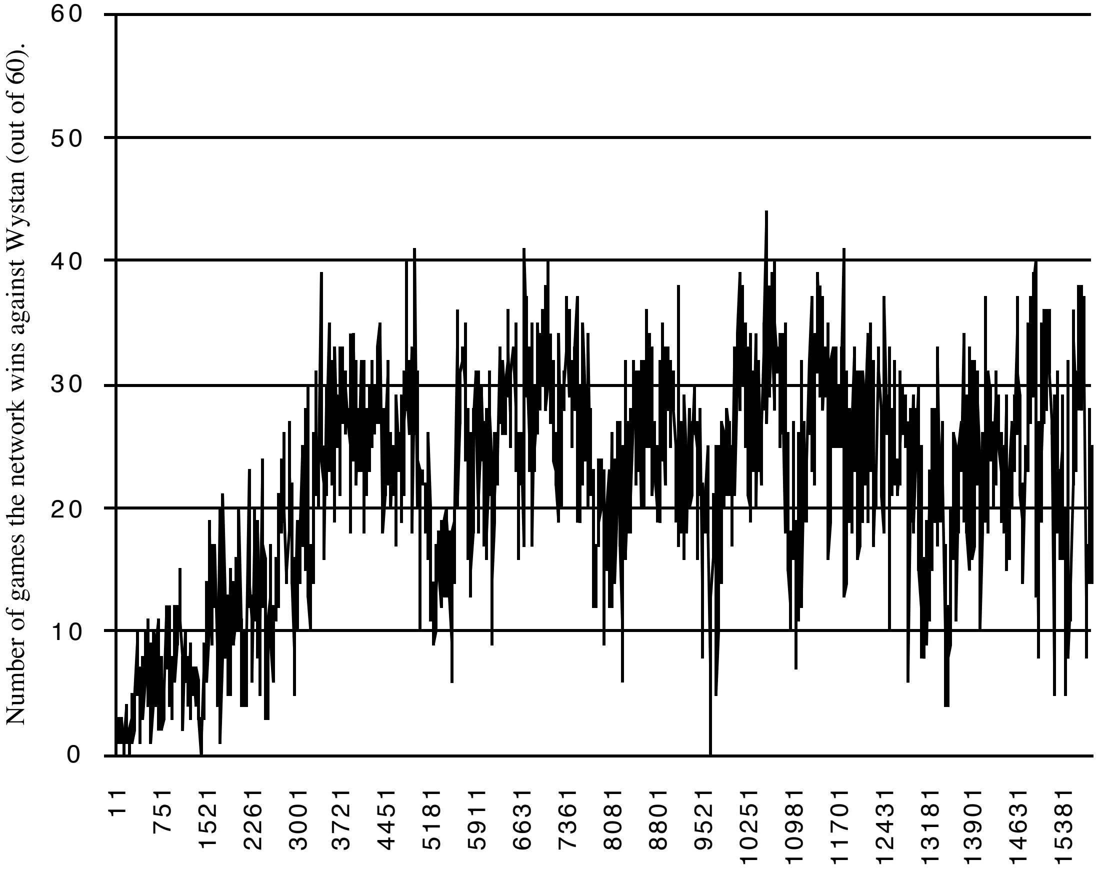
The weight updates were accumulated during the game and the actual update was performed after the game was over.

Squashing function brings the final reward on the same scale with the intermediate reward values. Also, the particular shape of the function enforces the condition that the more discs the program wins/loses at the end of the game, the less important it is to predict the exact number of discs. At the same time, without denominator of 10, the final reward would be nearly insensitive to the winning margin (Fig. 6).

Unlike backgammon, Othello is a deterministic game, so to ensure a sufficient exploration a stochastic factor was added into the learning process. During the search a better move was ignored versus already considered one with probability 0.1.

Originally the program was created as a player program for the game server, to allow it to be trained (and compete) against other Othello players. Due to this design and to simplify a weight updating procedure not one but two networks were trained competing against each other. It also allowed a unique opportunity to explore differences between white and black players.

Fig. 7. Performance of the network.



The program's performance was measured by running it against another Othello playing program, Wystan[2]. Wystan uses an evaluation function that incorporates several high-level features such as mobility and stability. The games were started from several different positions generated as follows: consider the initial board setup (Fig. 1). All combinations of the 4 first legal moves from this position will result in 244 different board states. Tossing away the boards that are equal with respect to rotation and reflection we are left

[2] Author Jeff Clouse. Personal communication.

with 60 unique positions. Both programs played 60 games starting from these positions. The number of games won by the network was recorded as the performance measure.

The network was trained for 10 games, then the learning was switched off and the network competed against Wystan as described above. Fig. 7 shows the learning curve obtained during nearly 15,000 training games.

There are two major observations following from this graph. First, the network is rather unstable, the performance is changing by a value of 10 or even more during just 10 training games. We see significant high-frequency oscillations on the graph. Second, the average performance of the network grows during the first 3,500 games, then it flattens and starts to oscillate. Let us forget the former observation for a moment and concentrate on the latter.

An analysis of the game of Othello shows that it has at least three rather distinguishable stages: opening-, middle-, and end-game. Every phase lasts approximately 20 moves and has a unique strategy. For example, mobility and stability play important roles during the middle-game, whereas the number of discs is the main factor during the end-game.

Training one network to evaluate board states in different game phases is the same as to train the network to perform different tasks at different times. The network can become a victim of the effect of temporal crosstalk [3]. It seems reasonable to have a different network trained for every game phase. This is the subject of the next section.

Application Coefficients

Three different networks were trained, one for each game stage. Instead of having three different evaluation functions, the transitions between stages are smoothed by application coefficients [1]. The disc count provides a good estimate of the game stage and is used as an argument in the following function:

$$AC_i(n) = Exp\left(-\frac{(n-\mu_i)^2}{\sigma^2}\right)$$

In the experiments σ=20, and μ assumes values 4, 34, and 64. Fig. 8 shows the plot of the AC(n) for these values.

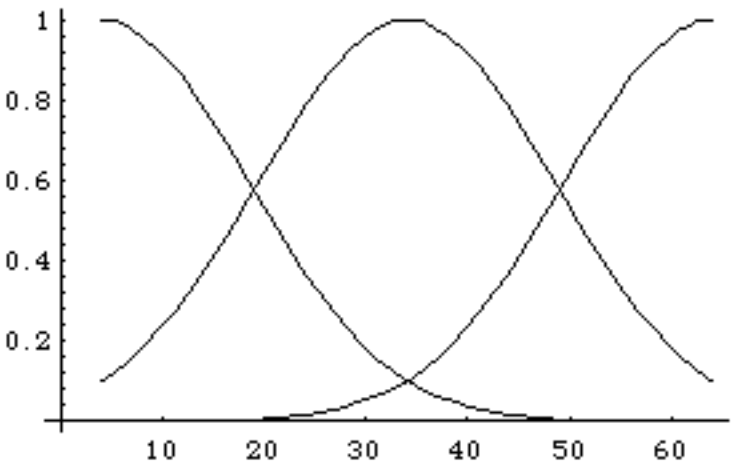


Fig. 8. AC(n) for σ=20, and μ = 4, 34, and 64.

The evaluation function will be

$$Eval(n, board) = \sum_{i=1}^{3} AC_i(n) Net_i(board),$$

where *$Net_i(board)$* is the output of the *i*th network given the current board state. A simple modification of the backpropagation formula is also required:

$$\Delta w_k^{(i)} = \alpha\left(t - \sum_i AC_i Net_i\right) AC_i f©\left(\sum_l w_l^{(i)} o_l^{(i)}\right) o_k^{(i)},$$

where *t* is the target output, $o_j^{(i)}$ is the output of the *j*th hidden unit in the *i*th network, $\Delta w_k^{(i)}$ is the change to be made to the weight from the *k*th to the output unit of the *i*th network, *f* is the activation function (squashing function in our case), and Net_i is defined as following:

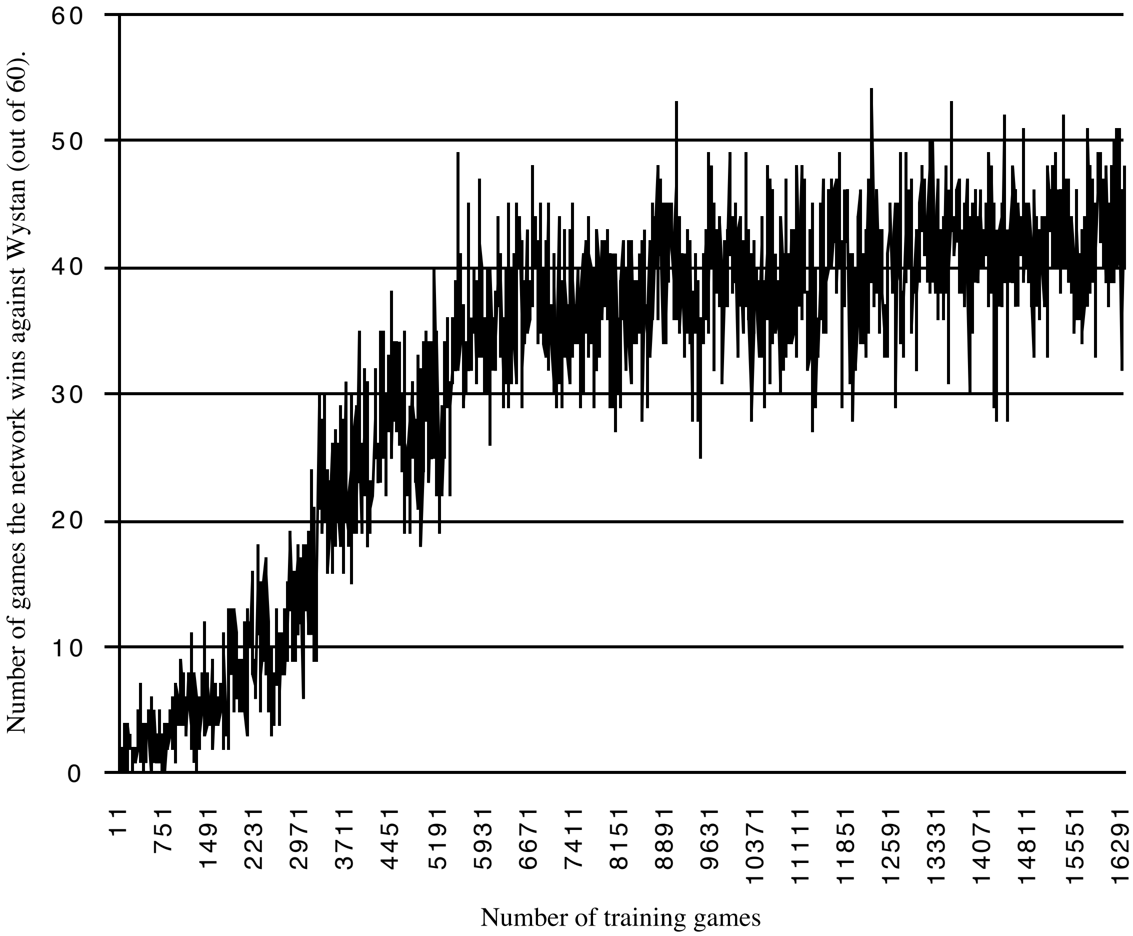
$$Net_i = f\left(\sum_j w_j^{(i)} o_j^{(i)}\right).$$

You may compare this with the conventional form:

$$\Delta w_k = \alpha(t - Net) f©\left(\sum_l w_l o_l\right) o_k$$

The training process was repeated for the modified version of the program, Fig. 9 shows the learning curve obtained during nearly 15,000 training games. The network achieves significantly better performance, displays less instability than the previous version, and the oscillations of the average performance disappeared.

Fig. 8. Performance of the network with application coefficients



6. Results

During the earlier exploration several network architectures were tested. The network that presented in the paper was the fastest to learn and achieved the best performance. The best recorded result (54 out of 60) shows that the network is capable of outplaying its opponent consistently. Remember that all possible openings of 4 moves were considered, when analysis of the expert play shows that at least one third of this set is unlikely to appear in the top-level matches. Experts agree on avoiding the so-called "parallel" opening (c4c5).

Note also, that at least one of the openings is definitely a no-win situation for the black (black has its disc in X-square).

As a non-beginner, I have not yet managed to beat the program. From watching the network's play, it certainly appears to "know" to avoid X-squares when necessary and at the same time the program may take X-square if it leads to its advantage. It definitely gained some concept of mobility and stability along the edges.

7. Future Work

Although the network showed rather good results the following points may be addressed to improve the network performance:

Othello may be considered as a three-color game in which the empty squares represent the third "color". The patterns of empty squares are nearly as important as patterns of the "normal" colors. The current version of the network attends to this problem by simply "ignoring" the squares that are empty. It seems rational to add an extra set of input units that are active when the corresponding squares on the Othello board are blank, and inactive when they are occupied.

The question of network capacity was not given its full consideration. The author has reason to believe that expanding the network by adding extra feature maps may boost the performance even further. An alternative would be to attempt to grow the network dynamically as it learns.

The application coefficients showed that it is a good mechanism for producing better evaluation functions, but the way they were introduced into the program is rather spontaneous and requires deeper investigation. The author thinks that adding some learning mechanism for the application coefficients may actually improve the accuracy of defining the game stage. Although J. Clouse [2] argues that using sigma-pi units in a two-layer network do not produce significant improvement, the other techniques (e.g. gating networks) should also be considered. For example, R. Jacobs [3, 4] presents a system composed of several different "expert" networks and a gating network that distributes training instances among experts. He also shows that the gating network is capable of *learning* how to make this allocation.

8. Conclusion

It has been shown that incorporating spatial and temporal characteristics of the domain into the network structure results in creating a more accurate evaluation function. The overall training process become faster and more stable. Relying on raw board information the neural network outperforms the fine-tuned algorithm that uses high-level features of the game.

Acknowledgements

I would like to thank Paul Utgoff for all his help. I am also grateful to Jeff Clouse and Neil Berkman for their valuable input and many stimulating conversations.

References

1. H. Berliner, "On the construction of evaluation functions for large domains", *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*. Tokyo, Japan: Morgan Kaufman, (1979).

2. J. Clouse, "Learning Application Coefficients with a Sigma-Pi Unit", Master Thesis, Department of Computer and Information Science, University of Massachusetts at Amherst (1992).

3. R. A. Jacobs, "Task Decomposition Through Competition in a Modular Connectionist Architecture", Ph.D. Thesis, Techical Report 90-44, Department of Computer and Information Science, University of Massachusetts at Amherst, (1990).

4. R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts", *Neural Computation* **3** (1991) 79-87.

5. Y. Le Cun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel, "Backpropagation applied to handwritten zip code recognition", *Neural Computation* **1** (1989) 541-551.

6. K.-F. Lee, S. Mahajan, "A Pattern Classification Approach to Evaluation Function Learning", *Artificial Intelligence* **36** (1988) 1-25.

7. K.-F. Lee, S. Mahajan, "The Development of a World Class Othello Program", *Artificial Intelligence* **43** (1990) 21-36.

8. D. Moriarty, R. Miikkulainen, "Evolving Complex Othello Strategies Using Marker-Based Genetic Encoding of Neural Networks", Technical Report AI93-206, Department of Computer Science, University of Texas at Austin (1993).

9. P. S. Rosenbloom, "A World-Championship-Level Othello Program", *Artificial Intelligence* **19** (1982) 279-320.

10. D. E. Rumelhart, G. E. Hilton, and R. J. Williams, "Learning internal representation by error propagation", In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. E. Rumelhart and J. L. McClelland, eds., Vol. I, 318-362. Bradford Books, Cambridge, MA.

11. R. Sutton, "Learning to predict by the methods of temporal differences", *Machine Learning* **3** (1988) 9-44.

12. G. Tesauro, "Practical issues in temporal difference learning", *Machine Learning* **8** (1992) 257-278.

13. G. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play", *Neural Computation* **6(2)** (1994) 215-219.

14. S. Walker, "Neural Neworks Playing the Game of Othello", Undergraduate Thesis, Department of ECE, University of Queensland.

15. D. Mitchell, "Using features to evaluate positions in experts' and novices' othello games", Master thesis, Evanston, IL: Department of Psychology, Northwestern University (1984).

16. The guide to the game of Othello.
World-Wide Web page at http://web.cs.ualberta.ca/ ~brock/ othello.html