A Neural Network that Learns to Play Five-in-a-Row

Bernd Freisleben
Department of Electrical Engineering and Computer Science (FB 12)
University of Siegen
Hölderlinstr. 3, D–57068 Siegen, Germany

Abstract

In this paper a neural network that learns to play the board game of five-in-a-row is presented. The basic idea of the approach is to let an appropriately designed network play a series of games against an opponent and use a reinforcement learning algorithm to train the network to evaluate the non-occupied board positions by rewarding good moves and penalizing bad moves. The performance of the proposed network is demonstrated by presenting experimental results.

1 Introduction

Reinforcement learning algorithms for neural networks are based on the availability of an external teacher who is able to indicate whether the network is in a desirable state or not, without supplying instructive information about how to reach desirable states. Usually, a scalar *reinforcement* signal is provided to evaluate the network behaviour in terms of success or failure, but the network itself is responsible for obtaining directional information about possible behavioural changes by probing the environment through the combined use of *trial and error* and *delayed reward* [5].

An interesting and challenging application domain of reinforcement learning are computer controlled board games [6], where the reinforcement signal ("won" or "lost") might only be generated after a long sequence of moves has been performed. In this case, the network must somehow be able to assign credit or blame individually to each action in the sequence of moves that led to the final outcome, i.e. it must provide a solution to the *temporal credit-assignment problem* [4].

Several approaches applying reinforcement learning techniques to game playing have been described in the literature. For example, the *temporal-difference learning method* [11] is employed in Samuel's early studies of machine learning using the game of checkers [9], in Schraudolph et al.'s investigation of GO [10], in Yee et al.'s tic-tac-toe proposal [17] and in Tesauro's backgammon network [15, 16] which outperforms his previous backpropagation-based Neurogammon system [12, 14] and is able to play at a master level that is extremely close to the world's best human players.

In this paper a neural network based on reinforcement learning is presented that learns to play *five-in-a-row*. The game, also known as *GO-MOKU*, is played on a 19 by 19 square mesh with two players, each of them having a supply of indistinguishable black or white pieces. The players take it in turns to play a piece on a board position. The winner is the first to complete a horizontal, vertical or diagonal line of five adjacent pieces of his or her colour.

The game has simple rules of play and is strategically far less complex than the related game of *GO*. Although it has been argued that the beginning player (black) is able to always win the game with his or her best moves [6], all attempts to develop a program exhibiting this skill have been proven to be unsuccessful. Previous approaches to automated five-in-a-row playing are based on more conventional AI techniques, such as game tree search methods and storage/analysis of significant situations to evaluate the importance of board positions [1, 3, 18], but a neural network does not appear to have been applied to the problem yet.

The approach proposed in this paper is based on developing an adequate network architecture to evaluate each non-occupied board position in order to determine the next move. The network is supposed to learn the evaluation function by playing a series of games against a software or human opponent and using the results to appropriately modify the connection weights in order to improve the performance. It will be shown that the trained network is able to achieve good results against a commercial five-in-a-row program and an experienced human player.

The paper is organized as follows. Section 2 presents the network topology designed to learn how to evaluate a particular board position and to decide which move should be made next. In section 3 the particular reinforcement learning algorithm used in this architecture is described. Section 4 illustrates the performance of the developed system by presenting experimental results. Section 5 concludes the paper and outlines areas for future research.

2 Network Architecture

The use of a neural network for playing five-in-a-row is motivated by the fact that a human player implicitly tries to recognize particular patterns of pieces on the board for which he or she remembers sequences of moves to succesfully win (or not loose) the game.

The basic idea of our approach is to design a neural architecture which allows to evaluate each non–occupied board position from two different viewpoints, the network's own viewpoint and the viewpoint of the opponent. This two–sided view is realized by a tree–shaped arrangement of two identically structured subnetworks, each with one output unit whose activation value determines the importance of placing a piece on the position currently being evaluated, as shown in Fig. 1. The two output units of the subnetworks (layer L_4) are connected to the output unit of the whole network (the root of the tree in layer L_5). The connections between L_4 and L_5

0-8186-7174-2/95 $04.00

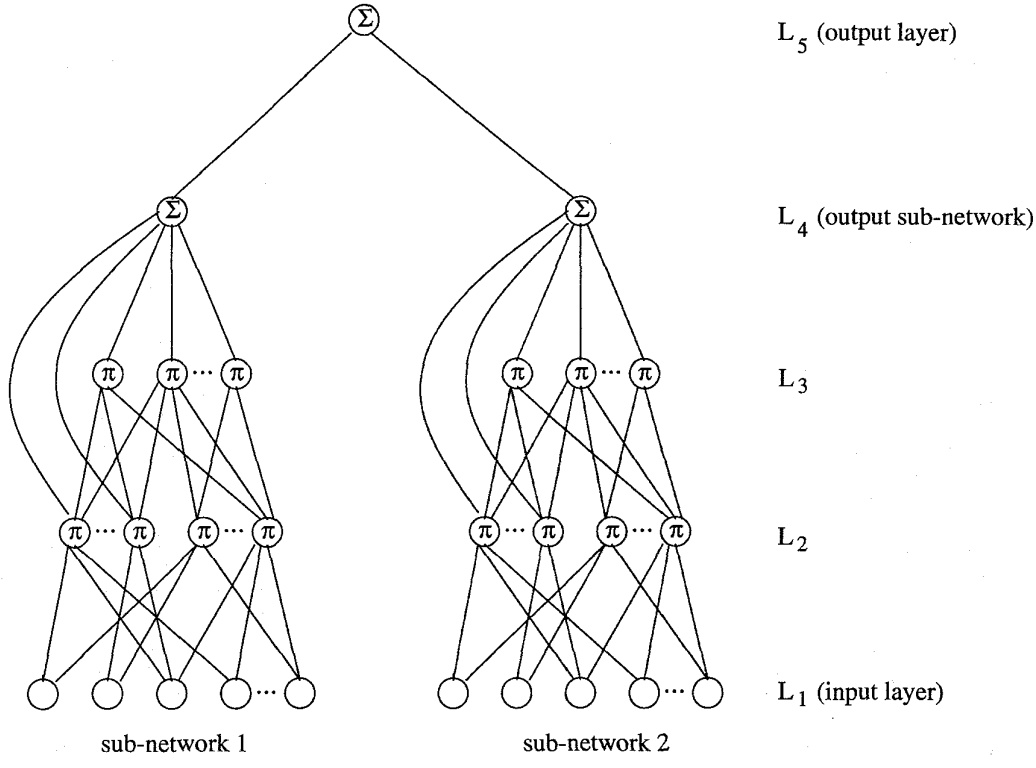

Figure 1: Network architecture for five-in-a-row

have fixed but possibly different weights to specify a bias towards attacking or defending play. A proper setting of these weights allows to avoid the quite often observable effect that in promising situations most human players are anxious to attack without considering defending actions anymore. The output unit of the network additively combines the weighted activations of the units in L_4 to determine the final network output of the board position currently investigated.

After all non-occupied positions have been evaluated in this way, the board position with the highest network output value is selected as the next move. If there are several equally high values, which is especially true in early stages of the game, one of them is chosen randomly.

Since the two subnetworks in the tree operate in an identical manner, it is sufficient to describe the functionality of only one of them. To evaluate a given, non–occupied board position (x, y), the 56 board positions surrounding (x, y), as shown in Fig. 2, are considered.

These 56 board positions contain all horizontal, vertical and diagonal rows with a distance of 4 pieces from (x, y) which may possibly lead to winning the game. Since such a star-shaped environment of (x, y) does not include all potentially winning situations, other positions around (x, y) are also considered. All board positions in this window–like environment of (x, y) are appropriately encoded, resulting in a vector where each component represents a particular position, with different values indicating if the position is empty (0.25), outside the border of the board (0.0) in case (x, y) is near the border, occupied by a piece of its own colour (1.0) or the colour

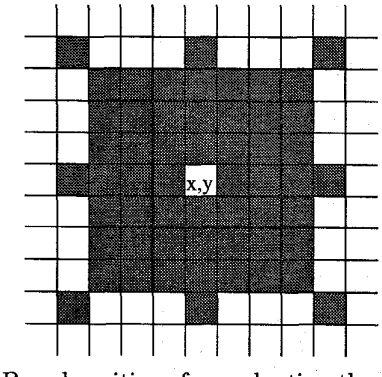

Figure 2: Board positions for evaluating the empty position (x, y)

of its opponent (0.0). For each non–occupied board position (x, y) there is one such vector which is presented to the input units of the subnetwork.

The input units have different numbers of connections to the units in layer L_2; these are so called Π-*units* [8], i.e. their outputs are computed as the weighted product of the connected inputs. Each of these 40 units has 4 or 5 connections to subsets of input units in order to reflect all possible combinations of horizontal, vertical or

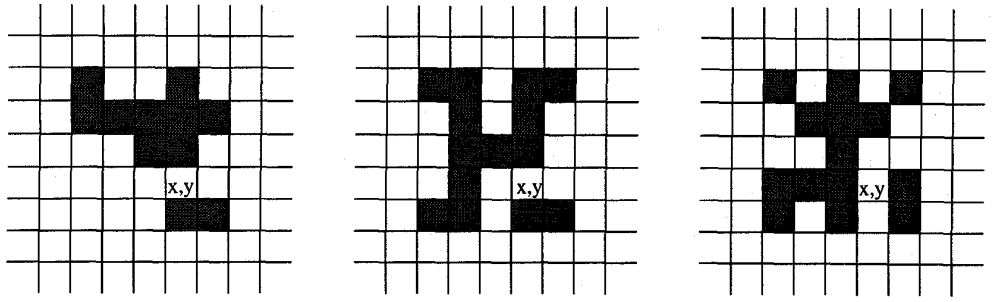

Figure 3: Other patterns considered

diagonal chains of five pieces of the same colour, depending on if (x, y) is part of the chain (4 connections) or not (5 connections). The units in L_2 are also asymmetrically connected to the 24 Π-units in layer L_3 to represent other important patterns which are formed by multiple chains of 5 pieces. Fig. 3 shows these three patterns which because of symmetries are present eight times in the environment of a board position to be evaluated.

These patterns are important, because placing a piece on (x, y) may not directly but later lead to a victory. The connections to the two layers of Π-units have fixed connection strengths of 1.0, because the network must be forced to recognize the importance of the associated pattern. Since in our coding scheme the board positions are represented by values between 0 and 1, a Π-unit with a larger number of connections (in L_3 there are up to 14 connections) would get a weaker activation than a Π-unit with fewer connections. To eliminate this effect, the activations are multiplied by a normalizing factor which is dependent on the number of connections.

The final layer of the subnetwork is the output layer with the single output unit described above. It has trainable connections to all units in the preceeding layer L_3 and fixed connections (1.0) to the part of the Π-units in layer L_2 which represents chains of five pieces (including (x, y)) in the *star–shaped* environment of (x, y). These fixed connection strengths are required, because the network must select (x, y) as the next move to win (or not loose) the game. In some sense, the connections with fixed connection strengths may be regarded as some sort of basic knowledge which is provided to the network about the game, similar to a novice human player to whom the rules of the game are explained. The fixed connection weights achieve that only those chains of pieces which allow to form 5 adjacent pieces of the own colour are considered important, since a piece of the opponent will through its coding of 0.0 yield an activation value of 0.0 in the corresponding Π-unit.

3 Learning to Play

Learning is aimed at optimizing the evaluations of board positions produced by the network. It is carried out as follows. First, the connection weights between L_3 and L_4 are initialized to random values and the network plays a series of games against a conventional, rule–based five-in-a-row program or a human player. Second, the moves of all of these games are recorded and then used as input vectors for the training phase. All games are then played again by the network alone. Since it is impossible to decide which moves are responsible for having won or lost a game, *all* moves of the *opponent* in the games which were lost by the network are considered, and with the current weights an evaluation for the present situation is produced. If the move determined by the network is different from that of the opponent, the move of the network is penalized and the move of the opponent is rewarded. This penalty–reward mechanism is realized by a variant of a particular reinforcement learning algorithm, *comparison training* [13], to modify the connection weights between L_3 and L_4. In order to do so, the difference between the network output and the move of the opponent is computed. The absolute value of this difference is distributed to the units in L_3 according to the error backpropagation procedure for Π-units described in [8] in order to determine the individual error δ_i produced by unit i in L_3. For the move of the opponent the error δ_i gets a positive sign, since the evaluation must be increased; for the move proposed by the network δ_i gets a negative sign. Finally, the weights w_i of the connections between L_3 and L_4 are modified according to

$$w_i(t+1) = w_i(t) + \alpha \cdot \delta_i(t) \cdot y(t) \cdot w_i(t) \qquad (1)$$

where $\alpha > 0$ is a learning parameter and $y(t)$ is the output of the unit in L_4.

In this way, the network learns the better moves of the winning opponent and can therefore improve its performance. In addition, not only the moves differing, but also the moves equal in both the network and the opponent are learned with a small positive error in order to reward an already correct move proposed by the network. This kind of reward is also applied to all moves of the games won by the network.

4 Implementation and Performance

The proposed network has been implemented in *C* on different machines (APOLLO DN 4000, SUN Sparcstation SLC, IBM RS 6000) [7]. In addition, a simple graphical user interface for playing five-in-a-row has been developed.

The performance of the network was tested in several series of games. Each of them consisted of two stages. In the first stage, a randomly initialized network played 50 games against another five-in-a-row program or a human opponent and the number of games won or lost were recorded. In the second stage, a total of 400 games was played against the opponent, and after each game the

corresponding moves were learned according to the procedure described in the previous section. Then the two stages were repeated for the next series of games. To perform both stages of a series of games about 6.5 hours were required on an IBM RS 6000 (30 minutes for stage 1, 6 hours for stage 2).

The first opponent was a simple five-in-a-row program which only considered the star-shaped environment of 4 pieces to evaluate a non-occupied board position. The network started with a score of winning 21 games out of 50, but increased the number of games won to 29 after two series of games. This kind of adaptation to the playing quality of the opponent could also be observed when a more suitably (but still randomly) initialized network performed originally better than its opponent. For example, the network decreased its playing quality from originally 30 games won to 25 after two series of games in which it was trained. This is not surprising, since the learning method used depends on the playing quality of the opponent. In the games where the network lost, the moves of the opponent are considered to be better than the moves performed by the network, such that the playing performance of the network is improved until the network has reached this performance. In the case where the training opponent is inferior, the network performance consequently decreases. This is the reason why in further tests where a trained network played against a copy of itself no significant improvements of the playing quality could be achieved, although the network resulting out of this series of games won 31 instead of 29 games against the originally used simple five-in-a-row program. However, there is no reason to assume that the network has learned something from itself; this effect is probably simply due to the random selection of positions with equally high evaluations. In a concluding test of the best trained network obtained in the experiments playing against a commercial five-in-a-row program, the network won 9 out of 10 games and also outperformed advanced human players by 7 to 3.

5 Conclusions

In this paper a neural network that is able to learn to play five-in-a-row was presented. An appropriate network architecture was developed to evaluate each non-occupied board position in order to determine the next move. The evaluation function was learned and improved by a reinforcement learning algorithm, a variant of comparison training, applied in a series of games played against another five-in-a-row program. In this learning method, the playing performance of the network is dependent on the opponent. It has been shown that the trained network is able to achieve good results against a commercial five-in-a-row program and an experienced human player.

There are several interesting areas for future research. First, the random selection procedure used among several alternatives with equally high evaluations must be analyzed, since an observation of the network's playing behaviour revealed some situations where the network obviously could have done better. Second, the environment of a non-occupied board position and the coding scheme used should be investigated in order to possibly extend the number of important patterns. Third, the inherent dependence on the quality of the opponent selected during the training phase should be reduced, but this would require developing a completely different approach to learning [2]. Finally, the pragmatic solution of the temporal credit assignment problem, i.e. making all moves of a game equally responsible for winning or loosing the game, is certainly improvable. However, this would require a detailed analysis of a large number of playing situations, and the important, highly influential moves are probably hard to identify.

References

[1] E.W. Elcock and A.M. Murray. Automatic Description and Recognition of Board Patterns in Go–Moku. In [6], pp. 291–303.

[2] S. Epstein. Towards an Ideal Trainer. *Machine Learning*, 15(3):251–277, 1994.

[3] N.V. Findler. Some New Approaches to Machine Learning. In [6], pp. 304–324.

[4] S. Haykin. *Neural Networks: A Comprehensive Foundation.* Macmillan, New York, 1994.

[5] J.A. Hertz, A. Krogh and R. Palmer. *Introduction to the Theory of Neural Computation.* Addison–Wesley, Reading, Massachusetts, 1991.

[6] D. Levy, Editor. *Computer Games I & II.* Springer-Verlag, Berlin, 1988.

[7] H. Luttermann. Development of a Neural Network that Learns to Play GO–MOKU (in German). Master's Thesis, Department of Computer Science, University of Darmstadt, Germany, 1991.

[8] J. L. McClelland and D.E. Rumelhart (Eds.). *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1 and 2. MIT Press, Cambridge, 1986.

[9] A.L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3:211–229, 1959.

[10] N.N. Schraudolph, P. Dayan and T.J. Sejnowski. Temporal Difference Learning of Position Evaluation in the Game of GO. In *Advances in Neural Information Processing Systems 6*, pp. 817–824, Morgan Kaufman Publishers, 1994.

[11] R.S. Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44, 1988.

[12] G. Tesauro and T.J. Sejnowski. A Parallel Network that Learns to Play Backgammon. *Artificial Intelligence*, 39:357–390, 1989.

[13] G. Tesauro. Connectionist Learning of Expert Preferences by Comparison Training. In *Advances in Neural Information Processing Systems 1*, pp. 99–106, Morgan Kaufman, 1989.

[14] G. Tesauro. Neurogammon Wins Computer Olympiad. *Neural Computation*, 1:321–323, 1990.

[15] G. Tesauro. Practical Issues in Temporal Difference Learning. *Machine Learning*, 8(3):257–277, 1992.

[16] G. Tesauro. Temporal Difference Learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995.

[17] R.C. Yee, S. Saxena, P. E. Utgoff and A.G. Barto. Explaining Temporal Differences to Create Useful Concepts for Evaluating States. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pp. 882–888, AAAI Press, 1990.

[18] T.U. Zahle. A Program of Master–Strength to Play Five–in–a–Row. In [6], pp. 325–338.