Move Ordering Using Neural Networks

Levente Kocsis, Jos Uiterwijk, and Jaap van den Herik

Department of Computer Science, Institute for Knowledge and Agent Technology, Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht, The Netherlands
{l.kocsis,uiterwijk,herik}@cs.unimaas.nl

Abstract. The efficiency of alpha-beta search algorithms heavily depends on the order in which the moves are examined. This paper focuses on using neural networks to estimate the likelihood of a move being the best in a certain position. The moves considered more likely to be the best are examined first. We selected Lines of Action as a testing ground. We investigated several schemes to encode the moves in a neural network. In the experiments, the best performance was obtained by using one output unit for each possible move of the game. The results indicate that our move-ordering approach can speed up the search with 20 to 50 percent compared with one of the best current alternatives, the history heuristic.

1 Introduction

Most game-playing programs nowadays use the alpha-beta algorithm. The efficiency of alpha-beta is closely related to the size of the expanded search tree and depends mainly on the order in which the moves are considered. If good moves are inspected first, the likelihood of cut-offs is increased, decreasing the size of the search tree.

For move ordering, the existing programs frequently rely on information gained from previous phases of the search. Major examples are a transposition table and the history heuristic [4]. Additionally, the original move-ordering techniques which are independent of the search process are still valid. In chess, for example, checking moves, captures and promotions are considered before 'silent' moves. However, the resultant move ordering only based on the latter group is somewhat poor, mainly because of the small number of move classes and the incidental relation with the board position.

It is a challenging idea to use learning methods to obtain a better search-independent move ordering. To achieve this, we trained a neural network to estimate the likelihood of a move being the best in a certain position. Subsequently, the estimated quality of the moves was used for move ordering.

A similar approach to ours is the chessmaps heuristic [1]. This heuristic is pattern based and uses a neural network to learn a relation between the control of the squares and the influence of a move. There are two main differences with our approach. First, in the chessmaps heuristic the neural network is trained to evaluate classes of moves, while our neural-network approach is tuned to

L. Monostori, J. Váncza, and M. Ali (Eds.): IEA/AIE 2001, LNAI 2070, pp. 45–50, 2001.

distinguish individual moves. Second, in our approach the target information for a neural network is the likelihood of the move being the best in a certain position, whereas in the chessmaps heuristic the target vector is the influence of the move on the squares of the board.

We use as test domain Lines of Action (LOA), a game suitable for pattern-recognition techniques [5]. LOA is one of the games played at the Mind Sports Olympiads and the last Computer Olympiad.

The article is organised as follows. Three move-encoding schemes are investigated in section 2. The experimental set-up for evaluating the efficiency of the move-encoding schemes and the resulting move ordering is described in section 3. The experimental results are given in section 4. Finally, section 5 presents the conclusions and some ideas for future research.

2 Three Move-Encoding Schemes

The evaluation functions involved in our experiments used a feed-forward neural network with a hidden layer consisting of 40 neurons. Both the output and the hidden neurons used a sigmoidal activation function. A board position was encoded by 65 input units. Each square was associated to a neuron (+1 for a black piece, −1 for a white one, 0 for an empty square) with an additional neuron indicating the side to move. The output neuron represents the minimax estimation of the position. The evaluation functions were trained using temporal difference learning. More details about the set of evaluation functions are described in [2].

The neural network used for move ordering has to be provided with the current board position and the legal moves in the position. The board position can be encoded in the input of the neural network in an identical way as described for evaluation functions. However, the method of how to encode the moves leads to a difficult choice. In the following we investigate three schemes of encoding the moves, and analyse the potential strengths and weaknesses of each of them.

The most natural move-encoding scheme is to use the input vector. The move is then additional to the board position. A move in LOA has two components: a 'from' part (denoting the stone to be moved) and a 'to' part (the destination of the move). For instance, we may assign 8 input units to the file and 8 to the rank of the 'from' part, and similarly 8 input units to the file and 8 to the rank of the 'to' part. For every move, four units corresponding to the 'from-file', 'from-rank', 'to-file' and 'to-rank' of the move are set to 1, the rest of the units are set to 0. Consequently, the size of the input layer is 97 (65 units for the board and 32 units for the move). In the output layer one single unit representing the estimated quality of the move under discussion can be used.

The time necessary to propagate the input vector through the neural network is similar to the time used by our evaluation functions. The neural network, when used for move ordering, has to inspect all legal moves in all internal nodes of the search tree, while the neural network for the evaluation function (the most time-consuming component so far) has to inspect only all leaf nodes. Since the ratio between the number of leaf nodes and the number of internal nodes is

significantly less than the branching factor (approximately 7 compared to 30), the gain from a better move ordering with this encoding scheme will most likely be lost by the overhead produced.

A second move-encoding scheme results from an analysis of the first one. The main problem with the previous encoding scheme is that with one activation of the neural network we estimate the quality of one move only. To make it faster, we should estimate the quality of all moves in a certain position simultaneously. However, encoding of all moves at once in the input vector of the neural network is not feasible. An alternative is to have separate output units for all possible moves, an output vector with 4096 (64x64) units. At first glance, the activation time is huge. However, we only have to compute the outputs corresponding to the legal moves of the input position. Consequently, the 'effective' size of the output vector is only the branching factor of the game. If we compare the number of effective weights for this encoding scheme with the number of weights of the evaluation function, we can observe that with the same hidden-layer size, the time necessary for move ordering in an internal node is not more than 1.5 times the time used by the evaluation function. Considering that the number of internal nodes is 7 times fewer than the number of leaf nodes, with identical hidden-layer sizes, the time overhead of this move ordering will be approximately 20%. If the size of the hidden layer is somewhat smaller, the overhead will be insignificant (<10%). A potential problem with this move-encoding scheme is that, since the number of 'real' weights is high, the generalisation during the learning phase is likely to be weak.

A third move-encoding scheme was designed to overcome the weakness of generalisation. This encoding scheme is similar to the second one, except that we now have one output unit for every 'from' square and one output unit for every 'to' square ($64 + 64 = 128$ units) instead of one output unit for every move. The estimation of a move will be the mean of the activation value of the output unit corresponding to its 'from' part and that of the output unit corresponding to its 'to' part. So, the training information for a certain move is generalised to the moves with the same origin or destination. To reduce the 'effective' size of the output layer, the same technique can be applied as in the previous case. Now we have to compute the activation in the 'from' half of the output vector corresponding to the stones of the player to move (approximately 10 units) and the activation in the 'to' part of the legal moves. If we compare this move-encoding scheme to the previous one, we observe that the 'real' size of the output is reduced from 4096 to 128 (enforcing some generalisation between the moves) and that the 'effective' size is increased from approximately 30 to approximately 40 (avoiding a significant increase of the overhead).

Summarising the analysis of the three move-encoding schemes, we expect that the first one has a significant time overhead, and the second one a poor generalisation. The third encoding scheme is expected to generalise better than the second one, but it is somewhat slower. Since the analysis do not lead to a clear conclusion, we investigated all three move-encoding schemes in our experiments.

3 Experimental Set-Up

We performed two experiments: (1) the neural networks are trained to estimate the quality of the moves, and (2) the efficiency of the move ordering based on the neural networks is evaluated.

During the first experiment the neural networks were trained to recognise whether a certain move is found to be the best by a search algorithm using the evaluation functions mentioned in section 2. Three types of pattern sets were generated: a *learning set* used to modify the weights of the neural networks, a *validation set* used for training-stopping criteria, and a *test set* used to evaluate the move-ordering performance in this experiment. A pattern in these sets consisted of a board position, the moves which are legal in the position, and the move found to be the best by the search algorithm within a certain depth. The learning set and validation set were generated using 2- and 3-ply deep searches, while the test set used 4- and 5-ply deep searches. The reason for a shallower search for the learning set was to be able to generate large sets. The validation set and the test set contained approximately 5000 board positions each. Moreover, these positions did not appear in the learning set. The size of the learning set varied.

As learning algorithm we used RPROP [3], known for its good generalisation. Like most of the supervised-learning algorithms for neural networks, RPROP also minimises the mean square of the error (i.e., the difference between the target value and the output value). In RPROP, however, the update of the weights uses only the sign of the derivatives, and not their size. The algorithm uses an individual adaptive learning rate for each weight, which makes the algorithm more robust with respect to initial learning rates.

During learning, the target value for moves 'known' to be the best was 1, otherwise 0. The error measure for evaluation and stopping criteria was the rate of not having the highest output activation value for the 'best' move. In each epoch the whole learning set was presented to the neural network updating the weights after each epoch. The error on the validation set was tested after each update. The training was stopped if the error on the validation set did not decrease after 100 epochs. For the three move-encoding schemes of section 2, the size of the hidden layer and the size of the learning set varied. Other parameters than these seemed to have less influence on the final performance.

In the second experiment the efficiency of the move ordering was evaluated using the neural networks that resulted from the first experiment. We compared the size of the search tree expanded by the move ordering described above with the size expanded using the history-heuristic move ordering. The size of the search trees is averaged over approximately 600 positions covering complete game sequences.

4 Experimental Results

In the first experiment, using the three types of pattern sets of section 3, we trained the neural networks with the three move-encoding schemes, varying the

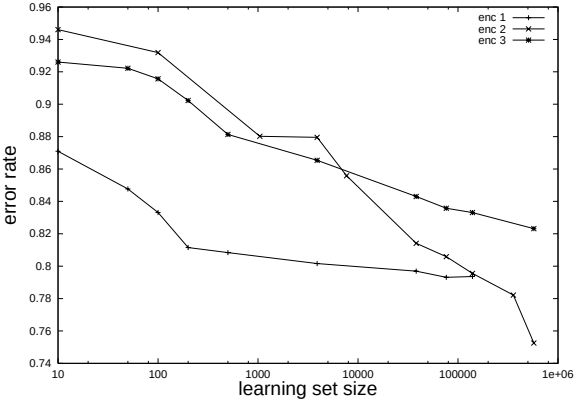

Fig. 1. The error rate on the test set as function of the size of the learning set.

size of the hidden layer and the size of the learning set. The increase of the hidden layer had only a clear influence on the first encoding scheme, namely, when large learning sets were used. Unfortunately, for this scheme the required time was already high with a small hidden layer. The remaining experiments, for each move-encoding scheme, were performed with a hidden layer consisting of 10 units and direct connections from the input to the output units. The dependency between the error rate on the test set and the size of the learning set is plotted in Figure 1. For smaller learning sets (up to 5000 board positions), the second encoding scheme had the worst results out of the three encoding schemes. However, this scheme gains the most from an increasing size of the learning set, clearly obtaining the lowest error rate for the largest training set. For smaller sets, the weak performance of the second encoding scheme was likely due to the poor generalisation (as predicted in section 2). For very large sets the available freedom of this scheme became more important than its generalisation. The first encoding scheme continuously outperformed the third one.

For the second experiment, we selected for each encoding scheme the neural network which had the smallest error rate in the first experiment. The three neural networks, corresponding to the three move-encoding schemes, were compared in terms of expanded search-tree size. The sizes of the search trees were normalised by dividing them by the sizes of the search trees expanded using the history heuristic for move ordering. The results are plotted in Figure 2. The node-count reduction can be interpreted as the reduction of the tree size if the history heuristic is replaced by the neural networks. The first and second move-encoding schemes seem to lead to similar results, and perform significantly better than the third scheme. Since the time used by the first encoding scheme to compute the estimated quality of the moves in a position is significantly higher than the time used by the other two, the second scheme can be considered as the best choice. Comparing the results obtained by the history heuristic, we observe a saving between 20 and 50 percent.

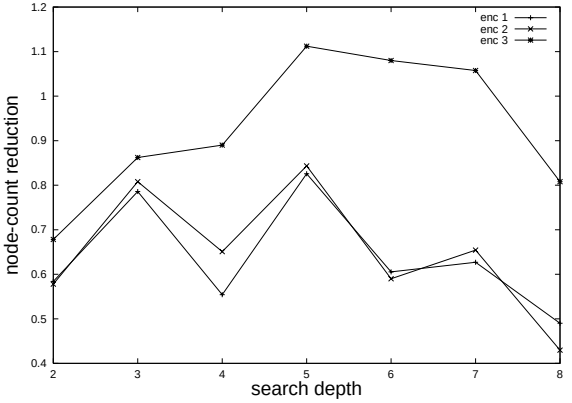

Fig. 2. The size of the search trees expanded using the best neural network for each encoding scheme as a function of search depth.

5 Conclusions and Future Research

This paper presents a new approach to move ordering, using neural networks to estimate the likelihood of a move being the best in a certain position. The move-ordering algorithm was tested for the game of LOA. From the experiments we conclude that the best choice is to use one output unit for each possible move of the game. The results from game-tree searches indicate that this approach to move ordering can speed up the search process with 20 to 50 percent compared with one of the best current alternatives, the history heuristic.

In practice, most search enhancements are not used in isolation. A major test for the move ordering presented is how it can be combined with other search enhancements or move-ordering techniques. Early results suggest that our move ordering works well in cooperation with a transposition table. An interesting issue is whether the neural networks obtained can be used in tree search for other purposes than move ordering. A challenging task is to develop selective-search techniques, which can exploit the strength of these neural networks.

References

1. Greer, K.: Computer Chess Move-Ordering Schemes using Move Influence. *Artificial Intelligence* **120** (2000) 235–250
2. Kocsis, L., Uiterwijk, J.W.H.M., Herik, H.J. van den: Learning Time Allocation using Neural Networks. *Working Notes of the Second International Conference on Computers and Games* (2000) 297–314
3. Riedmiller, M., Braun, H.: A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. *Proceedings of the IEEE International Conference on Neural Networks 1993 (ICNN 93)* (1993) 586–591
4. Schaeffer, J.: The History Heuristic. *ICCA Journal* **6**(3) (1983) 16–19
5. Winands, M.H.M.: Analysis and Implementation of Lines of Action. MSc thesis CS 00-03, Department of Computer Science, Universiteit Maastricht (2000)