Learning: An Effective Approach in Endgame Chess Board Evaluation

Mehdi Samadi , Zohreh Azimifar , Mansour Zolghadri Jahromi
Department of Computer Science and Engineering
Shiraz University, Shiraz, Iran
{samadi,azimifar,zolghadr}@cse.shirazu.ac.ir

Abstract

Classical chess engines exhaustively explore moving possibilities from a chess board position to decide what the next best move to play is. The main component of a chess engine is board evaluation function. In this article we present a new method to solve chess endgames optimally without using Brute-Force algorithms or endgame tables. We propose to use Artificial Neural Network to obtain better evaluation function for endgame positions. This method is specifically applied to three classical endgames: King-Bishop-Bishop-King, King-Rook-King, and King-Queen-King. The empirical results show that the proposed learning strategy is effective in wining against an opponent who offers its best survival defense using Nalimov database of best endgame moves.

1 Introduction

Nowadays, the promise and extraordinary potential of the Artificial Neural Networks (ANN) in solving various cognition problems have made them an appropriate approach for complex problem solving, such as the strategies required in game playing. Two-player games, such as chess, involve highly complex and non-linear intelligence, making the game particularly a good area to demonstrate the ANN as an approximation.

All today's sophisticated two-playing programs use some variants of the so called *alpha-beta* ($\alpha - \beta$) search algorithm. The name comes from a pruning technique which substantially reduces the expanded game tree and thus makes deeper searches feasible. The efficiency of $\alpha-\beta$ search algorithm depends heavily on its evaluation function. The game evaluation function is a key part in a chess engine. It is composed of a long list of parameters [3] describing the board positions and is obtained from extensive game experiences. Therefore, evaluation function improvement can help us to design more powerful chess engines.

The chess endgame is defined as the stage of the game in which there are few pieces left on the board. One of the objectives in chess engine studies is to play endgame stage optimally in terms of the number of moves. To meet this goal, chess engines employ endgame databases which store the number of moves required to force checkmate for all winning positions. The most commonly used database is known as Nalimov tableset including optimal[1] move for every endgame position (database size increases with the number of pieces). The database size, however, makes searching within the set timely very complex.

An alternative to reduce the chess engine time complexity is to aid the engine playing endgame positions suboptimally. The main idea is to employ ANN or other machine learning techniques such as Genetic Programming (GP) to solve chess endgames [1, 8, 10]. The suboptimality stems from the fact that the proposed engine will not look into a pre-defined table to copy the best move. The ANN is, however, trained according to features extracted from the board and it tries to obtain the optimum number of moves leading to the winning situation while the defender plays its best game strategy.

The main purpose of this work is to develop an ANN based on a previously defined set of board attributes. The ANN output is only one single value showing the optimum number of moves towards wining the game assuming the other side plays the best game. Therefore, it can be used as an evaluation function by the chess engine. We will explain how our chess learning strategy outperforms the existing techniques in terms of search tree time and space complexity. The simulation results will also show that best move prediction error is extremely improved compared to the state-of-the-art chess endgame evaluation functions.

2 Background Review

2.1 Neural Network

Human brain information processing method is the stem from which the concept of ANN originates. In the brain, it is

[1]The optimal move is the one that leads either the attacker to the quickest checkmate or the defender to the longest survival before checkmate or stalemate.

IEEE computer society

the network of neurons connected with axon and dendrites which makes learning possible [9]. The point of contact for neurons is called a synaps. ANN, too, has a network of neurons where elements and weights connections are processed. In the brain, the connections respond to axons and the weights respond to the synapses.

Stimulating human brain analytical function, the ANNs are capable of learning and recognizing non-linear and complex relationships. The experimental knowledge is received, stored and used by the ANNs in the shape of stable states or maps embedded in networks, and when responding to input variables, they are recalled. ANNs' capability of solving problems makes them suitable to lots of applications, as it was recently used in Artificial Intelligence (AI) fields.

Neurons have two functioning roles in an ANN: summing the weighted inputs from different connections and applying a transfer function to that sum [2]. The value taken this way is then sent to other neurons which are usually arranged in layers. There, the input layer receives the real world input and then forwards it, in the form of weighted output, to the next layer. In order for the ANNs to adjust the connection weights, ANNs should be trained by using a training algorithm and training dataset. The reader is referred to [9, 2] further reading.

2.2 Chess Engines

The search algorithm attempts to find the best move exploring all moves from a position. Using the $\alpha - \beta$ algorithm and an evaluation function, we try to compute the values of tree leaves.

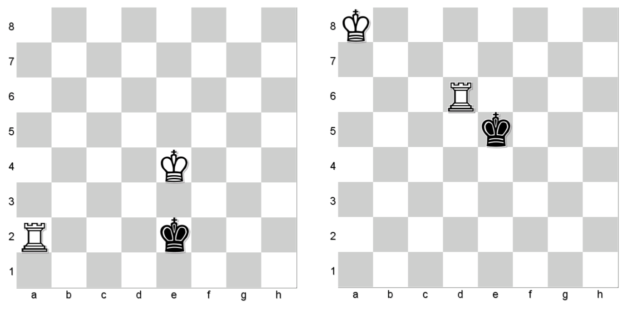
Finding and returning the maximum value (possible to be gained by one side of the game if another side performs the best move) can be achieved by the help of $\alpha - \beta$ pruning. When a leaf is reached by this algorithm, the algorithm evaluates the board position using its evaluation function. The found value, then, tries to find the best move.

The evaluation function applied at the game tree leaves, analyzes the position and by returning a numerical value indicates the player holding a better position. In endgames, due to expected and predictable limitations, we need stronger evaluation functions. Needless to say that human players play endgame much better than start and middle game.

3 Learning-Based Chess Engines

In this section we review some other machine learning techniques which have been used to solve endgame chess positions.

Si and Tang [10] attempted to solve King-Rook-King (KRK) endgame using ANN. Their objective was to predict the best next move for a particular board position. White king position, white rook and black king positions were considered as ANN input features. The proposed ANN approach was evaluated by normalized error measured only at trained stage. Although their method could significantly reduce training error, our experience with real game indicates that chess board position is not always an appropriate choice to train the ANN efficiently.



(a) black turn (b) white turn

Figure 1. Two instances of chess board position for KRK endgame.

Among various situations there are two major cases (Figure 1) in which changes in board position add more complexity to the ANN prediction function:

- Figure 1(a) shows that if we change the position of white rook to files *b*,*c*,*g* or *h* or if both kings are shifted left or right the optimal value (the value is 6 here) and the game strategy will not change. This is an example of a many-to-one function making the prediction complicated.

- In Figure 1(b) white should first secure its rook from being captured by black king, *e.g.,* moving it to file *a* or *h*, then using its king towards wining the game. Clearly, changing a rook position may drastically change both the optimal value and the next best move, *i.e.,* causing spikes in the ANN function.

A number of attempts ([6, 8]) have been made to use some board characteristics to improve learning efficiency of the ANN. Lassabo *et al* [8] proposed using GP to predict the next move. Although their approach is humanly understandable, a couple of issues must be addressed here. The proposed GP output is not the number of wining moves, but the next best move to be made. This makes the learning process more complex. According to the results reported from several real games, performance of the designed network is far from optimallity by a factor of ~ 2. Because of the associated large feature set this approach is also limited to a few endgame situations only. Further, the move ponder time is not efficiently utilized in the proposed method.

Some other works ([5, 10]) tried to mate the king opponent only; thus, the optimallity of the solution is not taken into consideration. These methods are more or less problem specific and are not expandable to other endgame positions. None of these techniques properly utilize the available time to improve the solution quality.

Our objective is, however, different than those reviewed above. We propose to design an ANN as an evaluation function. As stated before, the evaluation function plays a significant role in the performance of search algorithms for two-player games. A large body of research has been devoted to improve this function directly. The most powerful chess engines such as *"Deep Blue" [4]* and *"Crafty" [7]* explore the impact of the most recent progresses in evaluation function. *Therefore, we propose to use the idea of learning to improve the evaluation function.* Detail explanation comes next.

4 Methodology

One of the concerns with performance of the current engines is to improve their accuracy and time complexity. To address this issue, we propose to assign a fraction of the online search process to an offline estimation. In order to implement this idea, the offline process can be accomplished by exploiting some learning techniques.

Our idea of learning is to use some characteristics of board position as the input of the ANN which predicts the optimal value in terms of the number of moves for every instance. To obtain more accurate network inputs *i.e.,* board features, we consulted a chess grandmaster.

To simplify the description, this paper focuses on the simple $\alpha - \beta$ algorithm as the basic method used in 2-player game engines. This is for simplification only, though the actual implementation of the proposed method is done according to the latest improvement of $\alpha - \beta$ used by Crafty.

As a train set, the number of optimal moves to mate the opponent king is calculated by the search algorithm. After the training is efficiently completed, the trained ANN is used as an evaluation function in the search process. The limitation caused by space and time complexity of the search algorithm on one side and the accuracy of the evaluation function on the other side, are two main issues which are simultaneously improved by our proposed offline training approach.

This section describes the idea of learning in two steps: the proposed methodology and the ANN training strategies.

4.1 Construction of Training Data

To identify inputs of the ANN, we need to generate a number of endgame board instances. During the endgame, we start from the initial position and try to mate the black king in an optimal number of moves. In other words, during the play, returned value of the evaluation function decreases from initial value to zero (when it mates the opponent). Therefore, the evaluation function should predict the optimal value for all cases: hard, medium and easy ones. To achieve this, we should obtain our training data from all part of problem landscape. To obtain comprehensive train dataset from the entire problem domain we follow these steps:

- Generating some board positions in which white mates black (mate positions) and saves them in a database as goal states.
- Running Depth First Search (DFS) algorithm for one goal state selected randomly each time to any depth d. To avoid being trapped in some states while covering the entire domain, the following rules are taken into consideration:
  - Due to likely loops in the solution path, if we have already reached depth d, the optimal path is definitely shorter than d. Thus, from the beginning we choose d large enough to cover complicated states. To do this, one can start with small valued ds and monotonically increase to reach a maximum value which covers all cases.
  - A linear memory should be allocated to save all the previous states up to the current state. This prevents cycling in the current path.
  - At every state we choose the moving operator with minimum probability of occurrence to that point.

4.2 Endgame Feature Selection

In the particular case of chess endgame, the player usually follows specific algorithms or strategies. While involving few pieces, solving endgames is not a trivial task. Indeed, pieces involved have more freedom, thus, there are many possible moves per configuration. While chess engines usually perform a wide search through endgame tablesets to determine the correct solution, human players are able to exclusively perform a deep search to determine a wining strategy. This is a real advantage to solve endgames where planning an appropriate strategy is necessary. At this point we introduce some examples of endgame positions and describe the associated features to be used.

4.2.1 King-Rook-King (KRK)

In King-Rook-King (KRK) endgame, to avoid stalemate and to realize the check mate, it is inevitable to coordinate king and rook moves. To do so, players apply solving methods which are based on specific endgame patterns. In a KRK, it is obvious that the objective should be beating the defending king back to board edge (i.e., check mate). To obtain this result, the attacking king must be in direct opposition to the defending king and the rook must check the

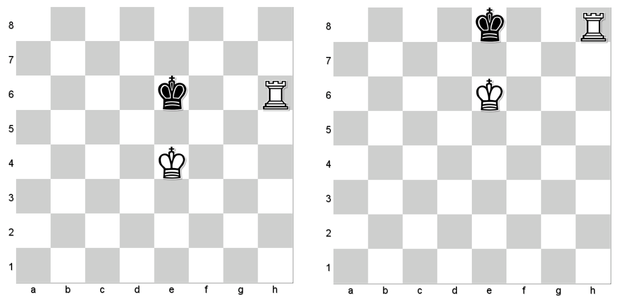


Figure 2. (a) Lateral check: there is opposition and the rook laterally checks the king. (b) Checkmate: the king can not defend anymore.

Table 1. Kings Characteristics - These features are commonly used by all endgame configurations.

Feature description	Value
The kings in opposition	[0-1]
The kings in diagonal opposition	[0-1]
The Kings in the same row	[0-3]
The kings in the small center of the board	[0-2]
The kings in the large center of the board	[0-2]
The black king in the edges of the board	[0-1]
The minimum distance between the kings	[0-6]

defending king on its side (Figure 2). This might need to be repeated several times.

In analyzing the board configurations to choose the best next move, simple patterns are used, though these simple patters may emerge too complex when combined. Meanwhile, the objective of mating the black king in the minimum number of moves is more difficult than, merely mating the black king and this necessitates knowing some patterns of predicting the best moves.

Table 1 summarizes all features describing both black and white kings positions which are used by all endgame configurations and Table 2 illustrates some characteristics with which KRK board positions are described. Each describing function gives an integer value which is the input for ANN. Chess Players usually do not consider all possibilities to make their next move. Players specific objectives is split up in board characteristics and can be achieved by specific moves.

4.2.2 King-Queen-King (KQK)

The combination of king and queen is to some extent similar to that of rook and king, but king and queen can mate the opponent faster. Because queen has more freedom of move and can protect diagonals, too, thus the black king cannot approach the queen in any direction.

Table 2. KRK Characteristics - Some of the characteristics are taken from [8].

Feature description	Value
The white rook blocks the black king	[0-1]
The white rook checks the black king	[0-1]
The white rook controls a line between the kings	[0-1]
The black king in a position capturing the white rook	[0-1]
The white king protects the white rook	[0-1]

Table 3. KQK characteristics.

Feature description	Value
The queen in the small center of the board	[0-1]
The queen in the large center of the board	[0-1]
The white queen in the adjacent row, column or diagonal of the black king	[0-3]
The black king in check	[0-3]
The queen in the eight adjacent squares of the black king and the white king supports it	[0-2]

The reason for choosing KQK endgame as another benchmark is that despite the fact that queen can mate black king faster, board patterns for KQK endgame are more complicated than those of KRK. In other words, the KRK domain is a subset of KQK situation when queen can move in all directions. According to our discussions with a chess grandmaster the characteristics describing this endgame position are defined and shown in Table 3.

4.2.3 King-Bishop-Bishop-King (KBBK)

Another benchmark with one more piece than the previous ones is two-bishop (KBBK) endgame. By pushing the opponent king into a corner we choose the easiest path to mate. In order to do this, bishops should cover side by side diagonals in a way that the black king cannot cross over and then the white king can push the black king into a corner. The opponent will be placed in a corner where it is squeezed in by bishops, tightly. When the black king reaches a corner, bishops have less attacking squares as shown in Figure 3(a).

Although it is believed that the only way it to mate with two bishops is by driving the opponent king into a corner, it is not always the case with mating. This can be done in a more sophisticated way: directing the king against an edge

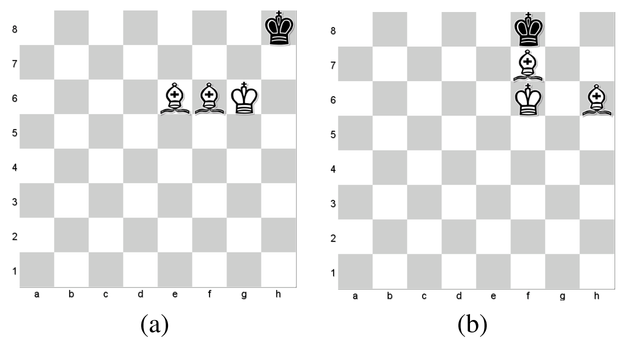


Figure 3. Two instance of chess board position for KBBK endgame. a)Bishops directing the black king to a corner, b)Bishops directing the king against an edge.

Table 4. KBBK characteristics - The light-squared bishop is called F1 and the black-squared one is called F2.

Feature description	Value
F1 and F2 check the black king	[0-2]
F1 or F2 in the eight adjacent square and the white king supports it	[0-2]
The bishops in the adjacent diagonals	[0-1]
F1 or F2 in the adjacent diagonals of the black king	[0-2]
The minimum distance between the black king and bishops located in same row or the same column	[0-6]

and delivering a mate (Figure 3(b)). This situation happens only if the opponent does not play optimally. Table 4 illustrates the board features for KBBK endgame[2].

5 Experimental Results

In this section we present the empirical results of using ANN as an evaluation function for endgame positions introduced in Sec. 4.2. The proposed method is examined in two aspects: ANN performance on any individual test data as well as ANN verification on real endgames. The performance of ANN is evaluated by Mean Square Error (MSE):

$$MSE = \frac{\Sigma_{i=0}^{N}(d_i - y_i)^2}{N} \tag{1}$$

where N shows the data size, y_i is the ANN output for the ith test data, and d_i is the desired output. The results are presented in Sec. 5.1.

[2]All the characteristic tables introduce above give only a portion of many features one could use to describe the board position.

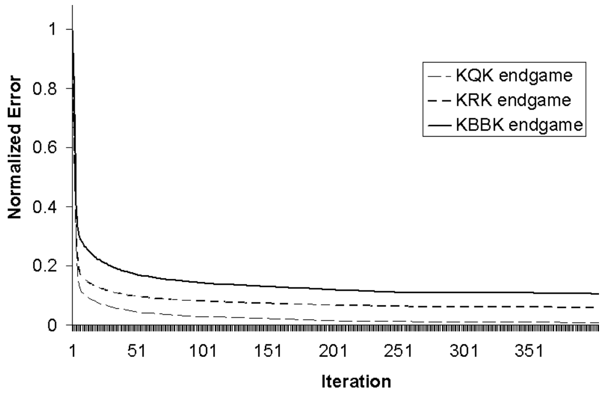


Figure 4. Training process of chess endgame back propagation algorithm.

Table 5. Mean-Squared error and standard deviation on train data.

End-Game	MSE	Standard Deviation
KQK	0.008347	0.021456
KRK	0.059514	0.021639
KBBK	0.116693	0.031931

Although, the MSE metric checks how well the ANN output fits to its desired value, experiments with heuristic searches indicate that a small MSE error does not necessarily result in a good game playing. To overcome this issue, the trained ANN was tested on 20 random real endgame situations with results presented in Sec. 5.2.

As discussed before, one advantage of the learning chess is that the trained ANN can be used either as the next best move predictor or as an evaluation function in the search tree. It will be explained that the quality of solution will improve if an ANN-based search tree is taken into consideration. In this section, the results for each endgame instance with different play time will be discussed, too.

5.1 Training

For each endgame introduced before, 10000 train data were generated using the algorithm presented in Sec. 4.1. The neural network used for KRK case is an all connected feed forward neural network with one hidden layer. The number of neurons in the hidden layer is five. There are 5-15 nodes in the input layer which define the characteristics for every position according to the tables given in Sec. 4.2.

For the purpose of this paper, only 10000 input-output train vector pairs were selected for each endgame. Figure 4 shows the training process with the error function defined in (1). All curves have been obtained by averaging over five times curve fitting. The plot displays result for each

Table 6. Comparison results of ANN-based engine with the existing techniques for different endgame instances. Clearly, Nalimov takes a large space to store all configurations.

Endgame	Engine Type	Time	Moves	Size
KQK	Crafty vs Nalimov	≤ 4 sec	9.8	NA
-	Nalimov vs Nalimov	NA	6.9	26×10^4
-	Learning vs Nalimov	≤ 0.15	7.6	$\leq$200
-	Learning vs Nalimov	≤ 0.30	7.4	$\leq$200
-	Learning vs Nalimov	≤ 1	7.3	$\leq$200
KRK	Crafty vs Nalimov	≤ 4	28.7	NA
-	Nalimov vs Nalimov	NA	13.5	26×10^4
-	Learning vs Nalimov	≤ 0.15	14.3	$\leq$200
-	Learning vs Nalimov	≤ 0.30	13.9	$\leq$200
-	Learning vs Nalimov	≤ 1	13.7	$\leq$200
KBBK	Crafty vs Nalimov	≤ 4	22.1	NA
-	Nalimov vs Nalimov	NA	14.4	16×10^6
-	Learning vs Nalimov	≤ 0.15	16.2	$\leq$200
-	Learning vs Nalimov	≤ 0.30	15.8	$\leq$200
-	Learning vs Nalimov	≤ 1	15.6	$\leq$200

endgame, separately. It is observed that for each case after about 400 iterations the training error reaches to the values presented by Table 5. The results indicate that the training stage converges very quickly to reasonable error values.

5.2 Verification

After the training stage, the proposed ANN was verified for real endgame situations. The trained ANN is used by the $\alpha - \beta$ as an evaluation function to evaluate chess board position. Therefore, the quality of solution depends on the tree depth in which the $\alpha-\beta$ goes thorough. It also depends on the total tree search time.

Each endgame was performed between an engine based on the $\alpha - \beta$ using the ANN as its evaluation function and another engine using the Nalimov tableset. As stated before, Nalimov tableset includes the best endgame moves for both sides, *i.e.,* the black side does its best play to survive. The number of moves to mate the black king in ``ANN v.s. Nalimov"game is compared with that of the ``Nalimov v.s. Nalimov "game (which indeed is the optimal value to solve an endgame instance). We also compared our results with those played by the referring chess engine, Crafty v.s. Nalimov.

The experimental results given in Table 6 were averaged over 20 random endgame positions. The time (second) assigned to each player to play the entire endgame, the averaged move number for white player to mate the black king, and the space requirement (byte) for each method are shown in the last three columns. The results indicates that the learning-based approach for KQK, KRK and KBBK outperforms Crafty 23%, 50% and 27%, respectively. Notice that the learning could also reduced the time by 96% efficiency. Evidently, the quality of solution improves when the engine has more time to decide. In other words, the significant saving in prediction time lets the engine make more appropriate decision at its turn by digging into the search tree.

6 Conclusions

In this paper we developed a new evaluation function using neural networks to solve chess endgame positions sub-optimally. The computed strategies were tested on three different endgames showing promising results. The results indicate that the proposed method can solve those endgames with significant improvements over the state-of-the-art techniques in terms of time, space, and move complexity.

Our future research direction is a two-folded plan. First, we make use of advances in the learning techniques to progress the efficiency of the evaluation function. We plan to improve the optimality of our MSE-based evaluation function in assigning the highest rank to the optimal moves. A combination of the MSE-based learning and other techniques such a the area under ROC curve will be studied. Second, the proposed method will be extended to other endgame positions which are harder for human to solve, *e.g.,* King-Knight-Bishop-King.

References

[1] M. Auton‘es, A. Beck, P. Camacho, N. Lassabe, H. Luga, and F. Scharffe. Evaluation of chess position by modular neural network generated by genetic algorithm. *EuroGP*, pages 1–10, 2004.

[2] R. Beale and T. Jackson. *Neural Computing: An Introduction.* Institute of Physics Publishing, Bristol, 1990.

[3] H. Berliner. Construction of evaluation function for large domains. *Artificial Intelligence*, 99:205–220, 1992.

[4] D. DeCoste. The signifcance of kasparov vs deep blue and the future of computer chess. *ICCA Journal*, (21):33–43, 1998.

[5] A. Hauptman and M. Sipper. Using genetic programming to evolve chess endgame players. In *Proceedings of the 8th European Conference on Genetic Programming*, pages 120–131. Springer, 2005.

[6] G. Haworth and M. Velliste. Chess endgames and neural networks. *ICCA journal*, 21(4):211–227, December 1998.

[7] R. Hyatt and M. Newborn. Crafty goes deep. *ICCA journal*, 20(2):79–86, 1997.

[8] N. Lassabe, S. Sanchez, H. Luga, and Y. Duthen. Genetically programmed strategies for chess endgame. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 831–838, New York, NY, USA, 2006. ACM Press.

[9] W. Mc Culloch and A. Pitts. A logical calculus of the ideas immanent in nervous activity, bull. mathem. *Biophys*, (5):115–133, 1943.

[10] J. Si and R. Tang. Trained neural network play chess endgames. *IJCNN*, 6:3730, 1999.