

Neural Networks Evolutionary Learning in Chess Game

Alberto Carrascal, Daniel Manrique, Juan Ríos, Claudio Rossi

Artificial Intelligence Department
Facultad de Informática; Univ. Politécnica de Madrid; Spain
jrios@fi.upm.es

Abstract. This paper proposes a new learning method based on evolutionary techniques to train artificial neural networks for playing chess. Instead of generating the following movement, artificial neural networks are proposed to accomplish the evaluation of each position generated by a search algorithm. A real-coded genetic algorithm combined with an improved version of the morphological crossover operator has been employed to train the neural networks. A self-play algorithm is applied to calculate the fitness of the individuals, which represent a set of weights and biases of a neural architecture.

1. Introduction

Most of chess-playing computer programs use highly optimized search algorithms, which generate a set of positions [1]. The evaluation of these positions is calculated by using a linear function, in which some features relative to the game are considered by means of a set of values [2]. Normally, these weighting values are arbitrary chosen, though there exist more elaborated approaches for their tuning, such as temporal differences [3], inductive inference [4] or genetic algorithms [5]. The possibilities of these kind of evaluation functions are very limited, since they only contemplate linear relations between the different features that describe a position.

Other researches convert the symbolic rules of the chess into numerical information that neural networks can learn [6]. In this kind of approaches, the neural network provides the next movement to be accomplished. Databases associated with already analyzed known positions provide the set of training patterns to be used with learning methods based on gradient descent [7]. The learning process consists on minimizing the differences between the movements returned by the neural network and the ones stored in the database. Since the number of possible movements to be made in a certain instant is very large, the use of these methods is limited to known positions. This is the case of the endgames. These supervised learning algorithms present two mayor disadvantages for this kind of problems. First, the high number of positions to analyse. Second, using subjective valuations for a certain position do not guarantee generating the most suitable movement. In order to avoid these problems, the use of unsupervised learning techniques such as genetic algorithms have been

employed on the game of checkers [8]. The disadvantage of this approach is the high number of input variables used: one per checkers square.

This paper presents a neural-genetic chess engine based on a neural network as a evaluation function that does not provide the next movement to be accomplished, but the valuation of the positions generated by an alpha-beta search algorithm. The genetic algorithm is based on a new crossover operator, called morphological crossover. Each individual in the genetic population codifies a set of weights and biases of a neural network that represents a chess player. The fitness of an individual is proportional to the number of games won against the other individuals of the population.

2. Neural Network as Evaluation Function

Using a neural network as evaluation function, instead of a linear function, offers a more flexible data processing associated to a chess position. Data processing comprises multiples non-linear relations among all analyzed chess position features. In this respect, an artificial neural network is able to learn more complex symbolic rules of the chess game.

Using an excessively complex neural network would be unviable due to the high computational cost of the training and execution processes since the number of position to evaluate is very large. In the results section, it is shown how it is possible to solve this problem using a generalized feed forward neural network with only one hidden layer. The neural architecture consists on ten input neurons (one per chess position feature), ten hidden neurons and one output neuron.

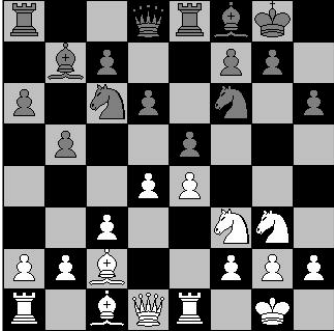
	<i>Features</i>	<i>Value</i>	<i>Features</i>	<i>Value</i>
	Stuff advantage	0	Isolated pawns	0
	Bishops pair	0	Passed pawns	0
	Doubled pawns	0	Center control	2
	Pawns advance	-2	King safety	0
	Stuff mobility	1		
	Stuff coordination	-1		

Fig. 1. Valued chess features for a given state of the chess game.

Aspects such like the stuff advantage, the pieces mobility, pawn structure or pair of bishops are numerically represented as shown in table 1. These values are standardized and given to the input neurons of the neural network. The stuff advantage fits tactics game considerations whereas the rest of features fit more

strategic chess game considerations. The output of the neural network is a numerical value that represents the position valuation. This value is used by the alpha-beta search algorithm to discriminate the possible movements to accomplish. Figure 1 shows on the right an example of the inputs received by the neural network for the state of the chessboard on the left.

Table 1. Computation of the characteristics of a chessboard state: p (pawn), b (bishop), r (rook), k (knight), q (queen) and k (king).

Chess feature	Formula to compute the chess feature
Staff advantage	$a_p \cdot (\#p_w - \#p_b) + a_b \cdot (\#b_w - \#b_b) + a_r \cdot (\#r_w - \#r_b) + a_k \cdot (\#k_w - \#k_b) + a_q \cdot (\#q_w - \#q_b)$ <p>where a_i are constant weights for each chess piece $\# \equiv$ number of pieces</p>
Bishop pair	<p>1, Two white opposite bishops and not two black opposite 0, Both or neither white and black two opposite bishops -1, Two black opposite bishops and two white opposite bishops</p>
Doubled pawns	$\#dp_w - \#dp_b$ <p>$\#dp \equiv$ Number of doubled pawns</p>
Pawns advance	$\sum_{i=1}^{\#p_b} J(p_{bi}) - \sum_{i=1}^{\#p_n} (9 - J(p_{ni}))$ <p>$J(p_i) \equiv i$ pawn row number</p>
Stuff mobility	Difference in the number of squares not dominated by the opponent that can be occupied by white pieces with respect to the black ones (bishop, knight, rook and queen)
Stuff coordination	Difference of the sum of the pieces protecting each white piece with respect to the black ones (except the king)
Isolated pawns	$\#ip_w - \#ip_b$ <p>$\#ip \equiv$ Number of isolated pawns</p>
Passed pawns	$\#pp_w - \#pp_b$ <p>$\#pp \equiv$ Number of passed pawns</p>
Center control	Difference of the sum of the pieces dominating each of the center squares by the white, with respect to the black
King safety	Difference of the sum of white pieces that can put the opponent's King into check with respect to the black (plus the castling)

3. Evolutionary Neural Network Training

Using genetic algorithms as neural network training method needs a previous codification stage. The figure 2 shows how the neural networks are codified by the individuals in the genetic algorithm.

The final real-numbered string obtained as result of the codification process contains the weights and biases of the neural architecture. This process is made from the upper to the lower neuron and from the input to the output layer. Finally, the biases are codified in the same way.

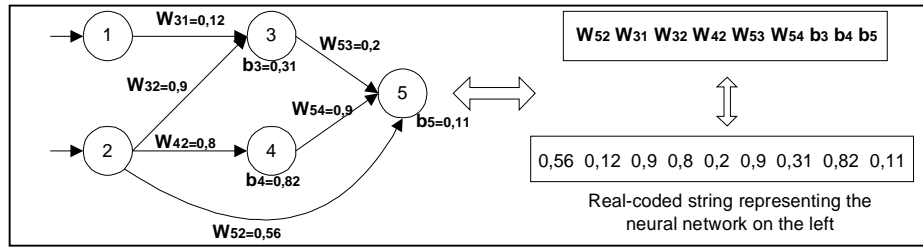


Fig. 2. Neural network codification

3.1 The Morphological Crossover for Chess Position Evaluation

Morphological crossover has been designed to solve general purpose optimization problems with real-coded genetic algorithms [9] [10]. This is the case when training artificial neural networks where weights and biases are coded by the individuals of the population.

Let $s \in D_{\mathfrak{R}}$ be a point in the search space, $D_{\mathfrak{R}}$, that comprises the set of all possible weights and biases of a given neural architecture. The neural configuration s is defined by the string $s = (a_0, a_1, \dots, a_{l-1})$, where $a_i \in \mathfrak{R}$. This operator works with each gene in the parents independently to obtain the corresponding gene in the two new neural configurations that are generated as a result of this operator. Let s_1, \dots, s_n be an odd number of neural configurations chosen from the actual population to be crossed, the n by l progenitors matrix is defined as:

$$G = \begin{pmatrix} a_{10} & a_{11} & \dots & a_{1l-1} \\ a_{20} & a_{21} & \dots & a_{2l-1} \\ \dots & \dots & \dots & \dots \\ a_{n0} & a_{n1} & \dots & a_{nl-1} \end{pmatrix} \quad \text{where } s_i = (a_{i0}, a_{i1}, \dots, a_{il-1}), i = 1, \dots, n.$$

The crossover operator works with each column $f_i = (a_{1i}, a_{2i}, \dots, a_{ni})$ in matrix G obtaining genes o_i and o'_i that belong to the two new neural configurations $o = (o_0, o_1, \dots, o_{l-1})$ and $o' = (o'_0, o'_1, \dots, o'_{l-1})$.

The procedure, adapted from the original morphological crossover, to generate the new sets of weights and biases $o, o' \in D_{\mathfrak{R}}$ is the following:

- a./ The morphological gradient operator, $g_b(f_i): D_f \rightarrow \mathfrak{R}$, is applied on each vector f_i , $i = 0, 1, \dots, l-1$, with a structuring element $b: D_b \rightarrow \mathfrak{R}$ defined as:

$$b(x) = 0, \forall x \in D_b, D_b = \{-E(n/2), \dots, 0, \dots, E(n/2)\},$$

where $E(x)$ the integer part of x ;

g_i is obtained as the value:

$$g_i = g_b(f_i) (E(n/2)+1) \quad i \in \{0, 1, \dots, l-1\}$$

If value g_i is high, that gene in the population is heterogeneous, while if it is low, that means that the values of that gene are converging.

- b./ Let be $\phi: \mathfrak{R} \rightarrow \mathfrak{R}$ the exploration / exploitation (EE) function that guides the genetic algorithm through the search space to the optimal solution. The maximum gene is defined as:

$$g_{\max} = \max(f_i) - \phi(g_i)$$

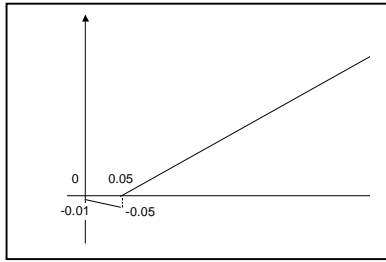
Likewise, the minimum gene is defined as:

$$g_{\min} = \min(f_i) + \phi(g_i)$$

Those values determine the crossover interval $C_i = [g_{\min}, g_{\max}]$, from where the desired value o_i is randomly taken. The i^{th} gene for the other descendant o'_i is obtained from inside the crossover interval using the following formula:

$$o'_i = g_{\max} + g_{\min} - o_i$$

The EE function allows to dynamically control the range of the crossover interval to avoid falling in local minima and to get a high convergence speed. When the individuals to be crossed are diverse (which implies a high value of the gradient) the crossover interval is made narrower according to the values $\max(f_i)$ and $\min(f_i)$, thus allowing to explore its interior searching for the optimum much faster. On the other hand, if the individuals to be crossed are very similar (gradient close to zero), which means that the population is converging, then it is advisable to expand the interval $[\min(f_i), \max(f_i)]$ to allow the exploration of new points in the domain, thus avoiding the possible convergence to a local optimum.



$$\phi(g_i) = \begin{cases} -(0.8 \cdot g_i) - 0.01 & \text{if } g_i \leq 0.05 \\ (0.421 \cdot g_i) - 0.021 & \text{otherwise} \end{cases}$$

Fig. 3. The EE function used by the evolutionary neural network training

The EE function employed by the evolutionary neural network training is shown in figure 3. This function is quite different from the one used in the original

morphological crossover. It has been designed to increase the exploitation capabilities of the genetic algorithm to get faster convergence speed in the neural networks training process. This way it is possible to evaluate more chess positions in less time to play better.

3.2 Fitness Evaluation and Replacement Criteria

Once the initial population of the genetic algorithm has been generated, the fitness of each individual is calculated. As the individuals represent different chess player minds, they are evaluated by playing a match with white color and other with black color against themselves. The result of these matches is accumulated, scoring +1 for each victory, + $\frac{1}{2}$ for draws and 0 for each defeat.

Each genetic algorithm iteration generates two new descendants as the result of applying the morphological crossover operator. The new offspring replaces the two worst individuals of the population using a SSGA criterion (Steady-State replacement Genetic Algorithm) [10]. The individuals of the population are evaluated again, not considering the previous fitness. This way, the new individuals compete in equal conditions with the rest of individuals.

4. Results

The neural genetic system proposed employs a genetic algorithm, with the morphological crossover, to train the artificial neural network executing one thousand genetic iterations. The figure 4 shows the mean score reached by the best individual in each genetic iteration after playing one match with white and black color with the rest of the individuals. It can be seen how the score is getting high, which represents that the artificial neural network is adapting, more adequately in each genetic iteration, to the chess game.

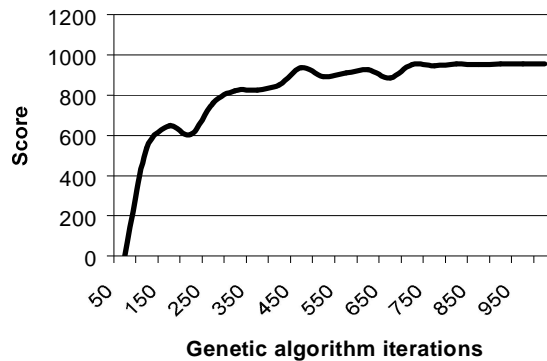


Fig. 4. Score evolution of the best players

When the genetic algorithm is converging, all the individuals of the population have very similar and high play level. In these conditions, the score stables, being 956 the mean value as shown in figure 4.

The neural-genetic-system has been compared with the AIChess program [5] (also based on genetic algorithms), playing 100 matches (50 with white color and 50 with black color). The results are shown in the table 2. Each row represents the percentage of matches won, drawn and lost respectively. The neural-genetic-system clearly outperforms AIChess. It is important to highlight that the number of won matches playing with black color is a 21% less that playing with white color in the case of the proposed system. This descent is 44% for the AIChess.

Table 2. Results of the proposed system against AIChess.

	White	Black
Win	76%	60%
Draw	14 %	22 %
Lose	10 %	18 %

5. Conclusions

This paper presents a new neural-genetic chess engine that employs artificial neural networks as evaluation function. The training process is unsupervised, so the skill of this system is gradually increased as the genetic algorithm converges. The results show that it is possible to improve the quality level of the engine by itself, playing the individuals against others. It is seen how it is possible to produce competent chess engines that exceed the expertise of the creator without using an expert knowledge. Artificial neural networks allow to use non-linear functions that improve the results. It is also possible to consider new relevant variables by simply adding new input neurons in order to obtain better performance. Using databases for openings and endgames improve the global performance of the neural-genetic system, but the research purpose is to show that this system reaches a satisfactory play level performance.

Acknowledgment

The work of the fourth author was funded by the Secretaría de Estado de Educación y Universidades of the Ministerio de Educación, Cultura y Deporte of Spain.

References

1. Heinz, E. A.: Scalable search in computer chess: algorithmic enhancements and experiments at high search depths. Friedrick Vieweg & Son/Morgan Kaufmann Publishers (2000).
2. Shannon, C. E.: Programming a digital computer for playing chess. *Philosophy Magazine*, 41, 256-275 (1950).
3. Baxter, J., Tridgell, A., Weaver, L.: Experiments in parameter learning using temporal differences. *ICCA Journal*, 21 (2), (1998) 84-99.
4. Jansen, A.R., Dowe, D. L., Farr, G. E.: Inductive inference of chess player strategy. *Pacific Rim International Conference on Artificial Intelligence*, (2000) 61-71.
5. Kendall, G., Whitwell, G.: An evolutionary approach for the tuning of a chess evaluation function using population dynamics. *Proceedings of the IEEE Congress on Evolutionary Computation Seoul, Korea* (2001) 27-30.
6. Posthoff, C., Schawelski, S., Schlosser, M.: Neural network learning in a chess endgame. *IEEE World Congress on Computational Intelligence, Orlando* (1994) 3420-3425.
7. Nunn, J.: Extracting information from endgame databases. *ICCA Journal*, 16 (4) (1993) 191-200.
8. Chellapilla, K., Fogel, D.: Evolving an expert checkers playing program without using human expertise. *IEEE Transactions on Evolutionary Computation*, Vol. 5, No. 4 (2001).
9. Barrios, D., Manrique, D., Porras, J., Ríos, J.: Real-coded genetic algorithms based on mathematical morphology. *3rd International Workshop on Statistical Techniques in Pattern Recognition, Alicante, Spain* (2000) 706-715.
10. Barrios, D., Carrascal, A., Manrique, D., Ríos, J.: ADANNET: automatic design of artificial neural networks by evolutionary techniques. *21st SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence, Cambridge, UK* (Dec 2001) 67-80.
11. Whitley, D., Starkweather, T.: GENITOR II: a distributed genetic algorithm. *journal of experimental and theoretical artificial intelligence*, (2) (1990) 189-214.